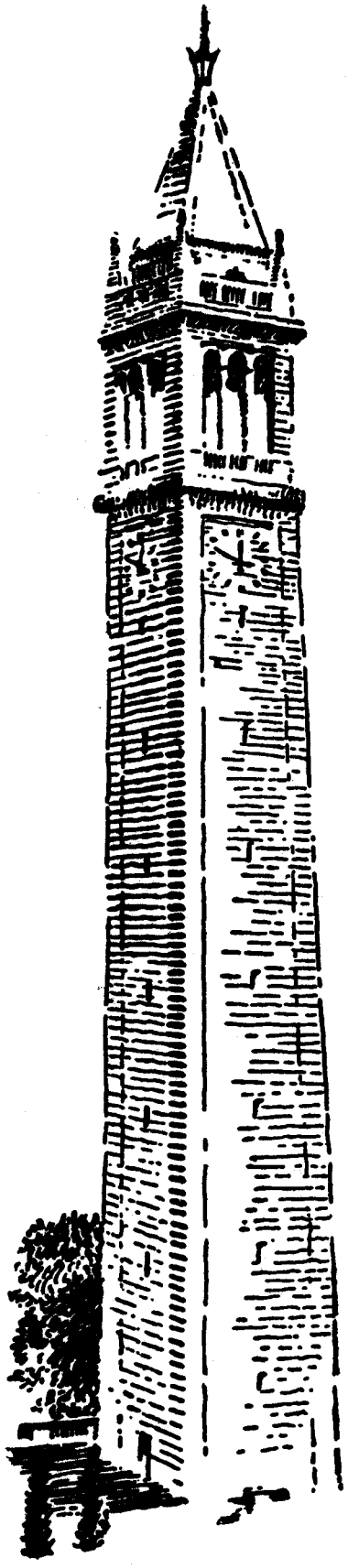

THE BCAM CONTROL AND MONITORING ENVIRONMENT

by

Bart J. Bombay

Memorandum No. UCB/ERL M92/113

11 September 1992

ELECTRONICS RESEARCH LABORATORY

THE BCAM CONTROL AND MONITORING ENVIRONMENT

by

Bart J. Bombay

Memorandum No. UCB/ERL M92/113

11 September 1992

THE BCAM CONTROL AND MONITORING ENVIRONMENT

by

Bart J. Bombay

Memorandum No. UCB/ERL M92/113

11 September 1992

THE BCAM CONTROL AND MONITORING ENVIRONMENT

by

Bart J. Bombay

Memorandum No. UCB/ERL M92/113

11 September 1992

THE BCAM CONTROL AND MONITORING ENVIRONMENT

by

Bart J. Bombay

Memorandum No. UCB/ERL M92/113

11 September 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

The BCAM Control and Monitoring Environment

Bart J. Bombay

September 11, 1992

Abstract

Accurate control and monitoring of manufacturing equipment is essential to integrated circuit production. Such a control scheme can take advantage of equipment models for the various steps involved in the production process. Unfortunately, the equipment used in integrated circuit manufacturing often changes with time and is always subject to various disturbances which in turn introduce significant fluctuation in performance. This report includes an adaptive regression model which evaluates itself and determines whether it should be corrected to better reflect equipment behavior. The model is modified through recursive estimation based on in-line wafer measurements. Decisions for model changes are based on formal statistical tests which use the principles of the regression control chart [1]. This strategy has been tested on the photolithography sequence in the Berkeley Microfabrication Laboratory [5][21]. In addition, a control scheme has been implemented which uses equipment models for feedback and feed-forward control of a manufacturing workcell. Included in this implementation are editing and generation functions for equipment setting recipes, a model editor, and interfaces to other Berkeley Computer-Aided Manufacturing (BCAM) applications. This implementation, named the BCAM Control and Monitoring Environment, is described in this thesis. Also included in this report are an instruction manual for use of the environment and a programming manual for further development of the code.

Table Of Contents

Acknowledgments v

Chapter 1 Introduction 1

Chapter 2 Theory of Discrete Process Control 3
2.1 An Introduction to Discrete Process Control 3
2.2 The Form of the Equipment Model 4
2.3 The Model Update Algorithm 5
2.4 Recipe Update Algorithm 11
2.5 Application of the Model Update and Recipe Update Algorithms 17

Chapter 3 BCAM Implementation 18
3.1 Introduction 18
3.2 The Ingres Database 19
3.3 The Role of X Windows in the BCAM Environment 20
3.4 Operations on Individual Machines 20
- The Recipe Editor
- The Model Editor
- Connections

3.5 Process Analysis Functions 21
3.5.1 Deduction of Process Order 21
3.5.2 Workcell Performance Prediction 22
3.5.3 Workcell Sensitivity Analyses 22
3.6 The Workcell Operations 22
- Basic Functions
- Workcell Controller

3.7 The Interface to Other BCAM Applications 23
- Alarm Generation
- Statistical Process Control
- Diagnosis
- Response Surface Plots

Chapter 4 BCAM Environment User's Manual 24
4.1 Introduction: Starting the BCAM System 24
4.2 The Equipment Window 25
4.2.1 General Description 25

4.2.2 The Equipment Recipe Menu 26
4.2.3 The Equipment Model Menu 26
4.2.4 The Equipment Window—Defining Equipment Connections 27
4.2.6 The Equipment Connections Menu 29
4.2.7 The Equipment Applications Menu 30
4.2.8 The Equipment Window—Editing Values 30
4.3 The Workcell Window 31
4.3.1 The Workcell Recipe Menu 32
4.3.2 The Workcell Wafer Menu 32
4.3.3 The Workcell Graph Menu 33
4.3.4 The Workcell Alarm Menu 33
4.3.5 The Workcell Options Menu 33

Chapter 5 BCAM Environment Programming Manual 34
5.1 Introduction: The Basic Program Structure 34
5.2 The Creation of the Main Windows 35
5.3 Naming Conventions 35
5.5 Structure Declarations 37
5.5.1 class machineClass 38
5.5.2 class modelClass 38
5.5.3 class processNode 39
5.5.4 class inputClass 40
5.5.5 class inputStepValueClass 40
5.5.6 class outputClass 41
5.5.7 class historyClass 42

Chapter 6 Conclusions & Future Work 43
6.1 Conclusions 43
6.2 Database Storage of Wafer Measurements 43
6.3 Alternate Storage Formats 43
6.4 Installation in Integrated Circuit Fabrication Facilities 44
6.5 Process Capability (Cpk) Evaluation and Process Simulation 44
6.5.1 Comparison of Control methods 44
6.5.2 Alarm Generation through Cpk Prediction 45

Appendix A Acronyms 46

Appendix B Symbols 47

Appendix C Adjoint Derivations of the Recipe Update Equations 50

Appendix D Ingres Table Formats 52

Appendix E Alphabetical Library Function Listing 60

Appendix F Description of the Source Files 70

Appendix G Principal Library Functions Listed by Hierarchy 75
G.1 The Beginning of the Program 75
G.2 Main BCAM Menu Actions 75
G.3 Activation of a Machine 76
G.4 Equipment Model Analyses 77
G.5 Workcell Operation 77

Appendix H Source Code Listings 80

Appendix I Known Bugs 81

Appendix J A G2 Formulation of Measurement Queuing Effects 82
J.1 Introduction 82
J.2 Methodology 83
J.3 Results 85
J.4 Conclusions 88
J.5 Future Work 89

References 90

Acknowledgments

I cannot begin to express the magnitude of my appreciation and gratitude to my research advisor, Professor Costas J. Spanos. His constant support and guidance have been invaluable to the success of my studies. I also thank Professor Seth Sanders for serving on my dissertation committee. In addition, I thank Dean David A. Hodges for his profound advice to me during both my undergraduate and graduate studies, and also for his support and leadership of CIM research at Berkeley. I am also most grateful to Professor Eugene Wong, Professor Martin Graham, Dr. Sheila Humphreys, Winsor Letton, Dr. Shahab Shiekholeslam, and David Wagner for their sage counsel through my years at Berkeley.

For their generous assistance in the proofreading of this thesis, I thank Dr. Shahab Shiekholeslam, Donald Zwakenberg, Sherry Lee, Eric Boskin, Sovarong Leang, and John Thomson.

I am also most appreciative of the efforts of Professor Gary May for his development of equipment model structures and his work on the BCAM LPCVD diagnostic system, Hao-Cheng Liu for his development of an equipment recipe editor, the BCAM diagnostic system and the database tables used by the BCAM Environment, Edward Wen for his work on statistical process control and response surface plotting, Sovarong Leang for his experimentation and development of photolithography equipment models and alarm generation algorithms, Sherry Lee for her work on the BCAM diagnostic system, Eric Braun for development of the BCAM response surface plotting application, Zhi-Min Ling for his development of microprocessing equipment models, and Lauren Massa-Lochridge for her plentiful assistance with the Ingres database software.

During my years at Berkeley, my participation on the 155 crew team had a profound effect upon my studies. The training assisted my development in all aspects of my life. At the heart of my rowing experience was my coach Jeff Wilk. His inspiration and spiritual

guidance gave me strength which allowed my studies to flourish. I must also acknowledge my teammates whose camaraderie and efforts contributed to my rowing experiences.

Also essential to the success of my research here at Berkeley have been the other members of the CIM/CAM groups at Berkeley: Professor Lawrence Rowe, Raymond Chen, Haifang Guo, Annika Rogers, Steven Smoot, Kwan Kim, Zeina Daoud, Mehdi Hosseini, and Soheila Bana.

I give my special thanks to Carol Block, Christopher Hylands, Ken Nishimura, Katalin Voros, Bob Hamilton, Genevieve Thiebaut, Heather Brown, Cheryl Craigwell, and the staff of the Berkeley Electronics Laboratory for their continual assistance in a multitude of matters during my stay at Berkeley.

For their unfailing support and knowledgable advice throughout my life, I am eternally grateful to my parents John and Barbara Bombay, my sister Helen Bombay, my grandparents Hunter and Annie Flores, and my cousins Michael Minatrea, Jeannie and Joseph Miller, John Minatrea, Richard Buford and Janine Minatrea, and the rest of my family. It is the great devotion and generosity of my family whom I must credit for all of my successes.

I thank the National Science Foundation for their support of my graduate studies at Berkeley. I thank also the Regents of the University of California, the Alumni Association of the University of California, Warren Dere, and Edward F. Kraft, and Cornell C. Meier for their support of my undergraduate studies.

This research has been jointly sponsored by the Semiconductor Research Corporation, the National Science Foundation, Texas Instruments, National Semiconductor, and the California MICRO program.

Chapter 1 Introduction

Until recently, IC fabrication facilities have relied mostly upon human experience to develop equipment recipes through trial and error. But as today's designs push the borders of existing technology, even the slightest maladjustment in equipment can drastically undercut production. The industry has therefore experienced extremely long start-up times when bringing a new product into regular production and costly cuts in production when maintaining or replacing manufacturing equipment. This problem is further compounded by frequent unanticipated changes in equipment performance. Today these changes are identified through Statistical Process Control (SPC), and sometimes a human operator attempts to re-adjust the process. Integrated circuit manufacturing technology can therefore reap great benefit from the application of a more advanced control system.

A control system which can meet the stringent demands of IC fabrication must be rather sophisticated. Simple feedback control applied to individual machines is unreliable due to the low inherent capabilities (the ratio of specification limit ranges to noise standard error) of the individual steps. A fabrication line control system must therefore have a solid base in statistical analysis of equipment performance. A control system must also be able to predict the equipment performance and adjust equipment settings whenever the predicted performance deviates from specifications. In order to establish this capability, a control system must use some sort of equipment models. Furthermore, in order to compensate for equipment changes, the models must allow themselves to be updated according to current manufacturing conditions. Statistically based models are desirable because analysis of equipment performance through regression techniques will allow the models to be updated efficiently.

The implementation of the control scheme includes regular checks of equipment models and recalculation of appropriate machine settings whenever those models are updated. This is the feedback portion of the controller. Whenever a process consists of several interdependent steps, e.g. the photolithography workcell, feed-forward control may be implemented to compensate early deviations in equipment performance by adjusting the settings of subsequent processing steps.

So that this control scheme may be integrated into the factory environment, it interacts with a database facility to store measurements and maintain a record of control recipes and equipment models. In addition, the control environment includes interfaces to alarm generation, diagnosis, and statistical process control applications.

The implementation of this control environment and the underlying analysis functions have been realized using C++ and X windows. This realization is named the Berkeley Computer Aided Manufacturing (BCAM) Control and Monitoring Environment and is described in this thesis.

After this introduction, the thesis devotes a chapter to the mathematical theory behind the controller's model update and recipe update algorithms. This is followed by a chapter describing the implementation of the BCAM environment. The BCAM user's manual and the BCAM programming manual constitute the next two chapters. Finally, the conclusions of this thesis are presented, and the possibilities for future development are discussed.

Chapter 2 Theory of Discrete Process Control

2.1 An Introduction to Discrete Process Control

The characterization of IC processes through equipment modeling has become a necessity in semiconductor manufacturing. Equipment models may be physical, empirical, or a combination thereof. Further, equipment-specific models are often updated to reflect the changing status of the equipment [7]. The Berkeley Computer-Aided Manufacturing (BCAM) group has developed several statistically based polynomial models that describe the behavior of some important IC manufacturing equipment: the Tylan low-pressure chemical vapor deposition (LPCVD) furnace, the Lam plasma etcher, and the photolithography workcell [8][9][10].

The equipment models described in this work consist of mathematical expressions that can predict the outcome of a manufacturing step (e.g. the thickness of the photoresist) given the settings of that step (e.g. spin speed, spin time, etc.). Such models are based on the statistical analysis of the results of designed experiments; manufacturing equipment is subjected to a well structured sequence of experimental recipes, and the resulting data is analyzed through stepwise linear regression. This leads to models which accurately reflect the operation of the equipment. The development of these models is assisted by a theoretical understanding of the physical behavior of the equipment [3].

The basic equipment model has been designed as a polynomial expression with a flexible representation so that it may operate efficiently within a comprehensive control system. Several input and output transformations (exponential, logarithm, roots, etc.) are also supported.

Since equipment characteristics change with time, a complete model structure must allow for updates of the model. This is accomplished by means of creating an adaptive

model which has two parts: an *original model* which represents the original state of the equipment, and a *correction model* which describes the deviation from that original state.

These models are used for performance prediction, the generation of descriptive response surfaces, and other control needs. Given its prediction capability, the model may also be used by optimization algorithms to deduce the required machine settings to meet target performance specifications. The optimization presented herein uses a multidimensional Newton-Raphson algorithm subject to inequality constraints on the machine controls.

2.2 The Form of the Equipment Model

Initially, an equipment model is derived using a designed experiment. This original model represents the general structure of the equipment behavior, i.e. the form of the polynomial terms used in the model equation; these terms represent the ways in which the machine's settings influence its outputs. For example, such a model has been developed for the photoresist spin-coat and bake equipment of the Berkeley Microfabrication Laboratory [3]. The four settings of the machine are spin speed x_1, spin time x_2, bake temperature x_3, and bake time x_4. The output is photoresist thickness z. For this example, a representative model is

$$z = c_0 + c_1 x_2 + c_2 x_4 + c_3 \frac{1}{\sqrt{x_1}} + c_4 \frac{1}{x_3 \sqrt{x_1}} + c_5 \frac{1}{x_1}. \quad (1)$$

where the symbols $c_0 \ldots c_5$ represent the coefficients of the terms of the model. Although the models are highly nonlinear with respect to the settings, they are linear with respect to the coefficients. This property allows us to use linear regression techniques to determine the coefficients which best fit the machine in question. All equipment models use this general format, although the number of terms (and the form of those terms) varies from machine to machine. Several transformations (such as the logarithm and the exponential)

are also supported on the inputs and outputs. As described next, the adaptive model always retains the basic structure defined by the terms of its original model, although the coefficients may be updated according to need.

2.3 The Model Update Algorithm

When, over time, a model fails to accurately represent a machine, corrections may be applied to the coefficients of the model. These corrections make up the correction model and are calculated by means of an update algorithm. The correction model in combination with the original model make up the complete adaptive model. The model update algorithm is initiated by means of a statistical process control alarm which is generated whenever the machine outputs differ significantly from those predicted by the model [5].

The model update algorithm is designed to modify the equipment model as necessary during routine equipment operation. Using historical records from a machine's operation, the update algorithm performs statistical regressions to determine the optimum correction model. This algorithm performs well in either a single product or a multi-product environment.

Because the adaptive model maintains a correction model while leaving the original model unchanged, it will never lose the information gained in the original designed experiment. For example, a correction model may evolve to compensate for a failing machine part; when that defective part is replaced, the adaptive model can quickly abandon the obsolete correction model and return to the original model.

Finally, each processing step has several outputs which can be controlled; hence, several models must be used to describe each step. The spin-coat operation, for example, is characterized by the thickness and the reflectance of the applied photoresist layer. Since each output is represented by a separate model equation, the update algorithm considers each of these outputs separately.

The model update algorithm is based on a weighted linear stepwise regression; it must therefore transform the historical records into a form suitable for such an analysis. This transformation procedure is as follows:

The values of the machine's past control settings are placed in the $\kappa \times n$ matrix[1] $\boldsymbol{X}$, each row of which contains the values of those settings corresponding to one run. Thus, for $k = 1 \ldots \kappa$ and $i = 1 \ldots n$, $x_{k,i}$ is the value of the i^{th} setting used for the k^{th} run. Since performance records become obsolete with time, a forgetting factor is applied in the weighting of the regression calculations in order to emphasize the most recent observations. In addition, a strict limit is placed on the number of observations to include in the model update calculations. This limit is called the *window size*, and is the maximum number of rows in $\boldsymbol{X}$. (When the number of data points available is less than the window size, then the number of rows in $\boldsymbol{X}$ is equal to the number of data points.) The appropriate choice of the window size depends on the rate at which machine performance is expected to drift and also the inherent capability of the machine.

The matrix $\boldsymbol{X}$ is then transformed into a $\kappa \times t$ matrix $\boldsymbol{T}$ that contains the respective values of the t model terms, as defined by the basic model structure. $\boldsymbol{T}$ has the same number of rows as $\boldsymbol{X}$, although it may have a different number of columns. For example, given the model described by equation (1), each row of $\boldsymbol{T}$ would have the form:

$$\boldsymbol{t}_k^T = \left[x_{k,2} \quad x_{k,4} \quad \frac{1}{\sqrt{x_{k,1}}} \quad \frac{1}{x_{k,3}\sqrt{x_{k,1}}} \quad \frac{1}{x_{k,1}} \right], \qquad k = 1 \ldots \kappa, \tag{2}$$

where $\left[x_{k,1} \quad x_{k,2} \quad x_{k,3} \quad x_{k,4} \right]$ is the corresponding row of $\boldsymbol{X}$ which contains the values of the machine settings used during the k^{th} run.

---

1. In this report bold-faced capital letters are used for matrices and bold faced lower case letters for column arrays. Row arrays are obtained by applying the transpose operator (^{T}) to a column array.

The update algorithm applies the *current model* (original model plus correction model) to the machine setting corresponding to each run and predicts the respective output values. It then takes the machine's corresponding historical output record z and subtracts from it the vector containing the predicted output values, thus yielding the *output discrepancy vector* Δz. The elements of Δz are defined by:

$$\Delta z_k = z_k - (t_k^T \cdot c + c_0), \qquad k = 1 \ldots \kappa, \qquad (3)$$

where c_0 is the current model's constant term coefficient, and c is the column vector of the current model's remaining term coefficients. This discrepancy vector will be used in order to determine the coefficients of the correction model.

The machine's performance records have been transformed into the term matrix T and the discrepancy vector Δz. These data, however, are not the result of a designed experiment, but rather the result of routine equipment operation; this fact will limit the number of correction coefficients which can be evaluated. If, for example, the machine has run with the same temperature setting throughout its relevant history, the data will not contain any information to determine if the effect of the temperature setting has changed. Similarly, if all the machine's settings have been held constant, then the correction model should only include a correction to the constant term coefficient.

In general, the data may support a correction to some, but not all, of the coefficients. In order to determine which coefficients can be corrected and how to correct them, a principal component transformation [6] is applied to the matrix T. This transformation has the added benefit that the transformed data is also orthogonally distributed—a property which greatly facilitates the subsequent stepwise regression.

However, before executing the principal component transformation, the terms matrix T must be properly numerically conditioned. This is accomplished by a transformation

which divides each column of $\boldsymbol{T}$ by the range of the corresponding term (defined as the difference of the maximum and the minimum values of the terms over the experimental space used to derive the original model):

$$\boldsymbol{V} = \boldsymbol{T} \cdot \boldsymbol{D}^{-1}, \tag{4}$$

where $\boldsymbol{D}$ is the $t \times t$ diagonal matrix whose nonzero elements are the ranges of the terms, and $\boldsymbol{V}$ is the $\kappa \times t$ normalized term array. This converts the terms into unitless numbers with comparable variances.

The principal component transformation starts with the evaluation of the weighted variance-covariance matrix of the data in the normalized terms matrix:

$$\boldsymbol{S_V} = \text{weighted covariance } (\boldsymbol{V}) = \frac{\boldsymbol{V_c^T} \cdot \boldsymbol{W} \cdot \boldsymbol{V_c}}{\boldsymbol{u^T} \cdot \boldsymbol{W} \cdot \boldsymbol{u}}, \tag{5}$$

where $\boldsymbol{S_V}$ is the $t \times t$ variance-covariance matrix, $\boldsymbol{W}$ is an $\kappa \times \kappa$ diagonal matrix containing the weighting coefficients $w_{kk}|_{k=1}^{\kappa}$ (off diagonal elements of $\boldsymbol{W}$ are zero), $\boldsymbol{u}$ is a t-dimensional vector whose elements are all ones, and $\boldsymbol{V_c}$ is the *centered* $\boldsymbol{V}$ array whose elements are given by:

$$v_{ckj} = v_{kj} - \bar{v}_{\bullet j}, \qquad k = 1 \ldots \kappa, j = 1 \ldots m, \tag{6}$$

where $\bar{v}_{\bullet j}$ is the weighted average of the elements in the j^{th} column of $\boldsymbol{V}$. This variance-covariance matrix is then factored:

$$\boldsymbol{B} \cdot \Lambda \cdot \boldsymbol{B}^{\mathrm{T}} = \boldsymbol{S_V}, \tag{7}$$

where Λ is the $t \times t$ diagonal matrix containing the t eigenvalues of $\boldsymbol{S_V}$, and $\boldsymbol{B}$ is the $t \times t$ orthonormal matrix[1] whose columns are the corresponding eigenvectors[2] of $\boldsymbol{S_V}$. The matrix $\boldsymbol{B}^{\mathrm{T}}$ is then used to transform $\boldsymbol{V}$ to its principal component space as follows:

$$\boldsymbol{V_{PC}} = \boldsymbol{V} \cdot \boldsymbol{B}, \tag{8}$$

where $\boldsymbol{V_{PC}}$ has the same dimensions, $n \times t$, as $\boldsymbol{V}$ and $\boldsymbol{T}$, and it contains the input terms data transformed into the principal component space. The output discrepancies may now be represented by rewriting (3) as:

$$\Delta z_k = z_k - (\boldsymbol{t_k^{\mathrm{T}}} \cdot \boldsymbol{D}^{-1} \cdot \boldsymbol{B} \cdot \boldsymbol{B}^{\mathrm{T}} \cdot \boldsymbol{D} \cdot \boldsymbol{c} + c_0) \qquad k = 1 \ldots \kappa, \tag{9}$$

or equivalently:

$$\Delta z_k = z_k - (\boldsymbol{v_{PC_k}^T} \cdot \gamma) - c_0, \qquad k = 1 \ldots \kappa, \tag{10}$$

where $\boldsymbol{v_{PC_k}^{\mathrm{T}}}$ are the rows of $\boldsymbol{V_{PC}}$, and $\gamma = \boldsymbol{B}^{\mathrm{T}} \cdot \boldsymbol{D} \cdot \boldsymbol{c}$ represents the vector of the term coefficients of the original model, transformed into the principal component space.

Next, a weighted stepwise regression is performed, considering each principal component separately in order to obtain a model *correction coefficient* $\Delta\gamma_l$ ($l = 1 \ldots t$) for that component. Only principal component directions showing significant variance should be considered; therefore, the algorithm examines only the principal components whose corresponding eigenvalues λ_l of $\boldsymbol{S_V}$ satisfy the inequality

$$\lambda_l > \frac{a^2 \cdot r^2}{\gamma_l^2}, \qquad l = 1 \ldots t, \tag{11}$$

where r is the magnitude of the range of possible values that the output may take, and a is a unitless empirical quantity taken to be 10^{-2} for the BCAM application. (The factors in

1. An orthonormal matrix has orthogonal columns (and rows), each of which has a magnitude of one. The inverse of an orthonormal matrix is simply its transpose.

2. Because $\boldsymbol{S_V}$ is symmetric positive definite, this factorization is equivalent to the singular value decomposition of $\boldsymbol{S_V}$.

(11) are necessary to ensure that both sides of the inequality have compatible units.) For each considered correction coefficient $\Delta\gamma_l$, a p-value[1] is calculated. If this p-value is low enough, the calculated value for that correction coefficient is accepted. Otherwise the correction coefficient is set to zero. The regression analysis takes the form of the system

$$\Delta\gamma = [V_{PC}^T \cdot W \cdot V_{PC}]^{-1} \cdot V_{PC}^T \cdot W \cdot \Delta z, \tag{12}$$

where $\Delta\gamma$ is the column vector of model correction coefficients in the principal component space. The significance of each correction coefficient is established by looking at the variance of each estimator. This variance is related to the standard deviation, σ, of the machine output:

$$\text{var}(\Delta\gamma) = [V_{PC}^T \cdot W \cdot V_{PC}]^{-2} [V_{PC}^T \cdot W^2 \cdot V_{PC}] \cdot \boldsymbol{u} \cdot \sigma^2. \tag{13}$$

The standard deviation σ is estimated from the residuals of the regression equation during the creation of the original machine model. Additional estimates of σ might be obtained during replicated runs.

In order to ensure the stability of the algorithm, an additional test is applied to prevent extreme corrections to the model. During an update procedure, no model coefficient may change by more than 60% of its previous value.

Once all significant principal components have been examined, the terms array V_{PC} is multiplied by the new correction coefficients $\Delta\gamma_l$, and the resulting vector is subtracted from the output discrepancy vector Δz. The weighted average of the elements of the resulting vector, if significant, is the constant term correction coefficient

1. The p-value is the probability of obtaining an estimate of the correction coefficient whose magnitude is greater than the considered estimate, assuming that the true value of the correction coefficient is zero. Typically, if the p-value is less than 0.05, then the correction coefficient is accepted as significant.

$$\Delta c_0 = \frac{\sum_{k=1}^{\kappa} w_{kk} [\Delta z_k - (v_{PC_k} \cdot \Delta \gamma)]}{\sum_{k=1}^{\kappa} w_{kk}}, \qquad k = 1 \ldots \kappa, \quad (14)$$

where the w_{kk} are the diagonal elements of $\boldsymbol{W}$. The correction coefficients are then put through the inverse transforms which bring them back into the original terms space, where they become the correction model's term coefficients

$$\Delta \boldsymbol{c} = \boldsymbol{D}^{-1} \cdot \boldsymbol{B} \cdot \Delta \gamma. \quad (15)$$

Finally the updated model coefficients are $\boldsymbol{c} + \Delta \boldsymbol{c}$ and $c_0 + \Delta c_0$, and the model update procedure is complete.

2.4 Recipe Update Algorithm[1]

The implementation of a feedback control system to integrated circuit manufacturing requires that the controller be able to update machine settings whenever equipment models change. The implementation of feed-forward control similarly requires the calculation of machine settings. For these purposes, a settings recipe update algorithm is required.

The unconstrained recipe calculation problem reduces to the following:

Solve for $\boldsymbol{x}$ such that

$$\boldsymbol{f}(\boldsymbol{x}) \cong \hat{\boldsymbol{z}}, \quad (16)$$

where $\boldsymbol{x} \in X \subset \Re^n$, the n-dimensional input space, $\hat{\boldsymbol{z}} \in Z \subset \Re^m$, the m-dimensional output space, and $\boldsymbol{f} : X \rightarrow Z$. An iterative algorithm is presented which starts with an

1. Note that some symbols in this section do not correspond directly with those of the previous section.

initial x_0 and generates a sequence $x_0, x_1, x_2, \ldots$ converging to the best compromise solution $\hat{x}$. Convergence properties of this method are discussed in [18].

Denote the j[th] component of f: as $f^j: X \to \Re$. Then at each iteration k, f can be linearized[1] about x_k to get

$$f(x) \cong f(x_k) + A_k \cdot (x - x_k), \tag{17}$$

where

$$A_k = \frac{\partial f}{\partial x}(x_k) = \begin{bmatrix} \frac{\partial}{\partial x} f^1(x_k) \\ \frac{\partial}{\partial x} f^2(x_k) \\ \frac{\partial}{\partial x} f^3(x_k) \\ \cdots \\ \frac{\partial}{\partial x} f^m(x_k) \end{bmatrix}_{m \times n} . \tag{18}$$

This leads to the modified problem: find a compromise solution x_{k+1} such that

$$f(x_k) + A_k \cdot (x_{k+1} - x_k) \cong \hat{z}, \tag{19}$$

If $n \geq m$ and $A_k A_k^T$ is invertible, then a solution[2] to this modified problem is

$$x_{k+1} = x_k + \Delta x_k = x_k - A_k^T [A_k A_k^T]^{-1} [f(x_k) - \hat{z}] . \tag{20}$$

Using a Euclidean norm, this solution is as close as possible to the previous solution x_k. If $n < m$ and $A_k^T A_k$ is invertible, then the least square error solution to the modified problem is

1. The models used in the BCAM recipe generation algorithm are such that the solution to the modified problem (19) is at each iteration close enough to the best compromise solution $\hat{x}$, so that the sequence of solutions $x_0, x_1, x_2, \ldots$ converges to $\hat{x}$. In general, the convergence properties of this method require bounds on the minimum and maximum singular values of the derivative matrix $\frac{\partial f}{\partial x}(\bullet)$ over X.

2. See Appendix C for a derivation of these equations.

$$x_{k+1} = x_k + \Delta x_k = x_k - [A_k^T A_k]^{-1} A_k^T [f(x_k) - \hat{z}] . \tag{21}$$

Equation (21) is equivalent to the solution produced by a Local Newton optimization method using the cost function $G_k : X \to \Re$, where

$$G_k(x) \equiv \frac{1}{2} \sum_{j=1}^{m} [g_k^j(x) - \hat{z}^j]^2, \tag{22}$$

$\hat{z}^j$ is the jth component of $\hat{z}$, and

$$g_k{}^j(x) \equiv f^j(x_k) + \frac{\partial}{\partial x} f^j(x_k) \cdot (x - x_k) \equiv f^j(x) \tag{23}$$

(i.e. the linearization of f^j about x_k). Equation (20) similarly results from the problem

$$\min \{ \|x - x_k\|^2 \mid G_k(x) = 0, x \in X \subset \Re^n \} . \tag{24}$$

Of course any practical implementation of this algorithm must not only require invertibility of the relevant matrices, but also put finite limits on the conditioning of those matrices.

The cost function G_k is simply the sum of squares of the deviations from the target values $\hat{z}^j$, $j = 1 \ldots m$. Different scales of measure for the target values $\hat{z}^j$ and the recipe values x^j can be accommodated by placing scaling factors into the cost function G_k to obtain

$$\tilde{G}_k(y) \equiv \frac{1}{2} \sum_{j=1}^{m} \{ \frac{1}{s^j} [g_k^j(Ry) - \hat{z}^j] \}^2 = \frac{1}{2} \sum_{j=1}^{m} \{ \frac{1}{s^j} [\tilde{g}_k^j(y) - \hat{z}^j] \}^2, \tag{25}$$

where the s^j ($j = 1...m$) scale the target values[1] and $\boldsymbol{R}$ is a diagonal $n \times n$ matrix containing the scaling factors for the recipe values[2] so that $\boldsymbol{y} \equiv \boldsymbol{R}^{-1}\boldsymbol{x}$. Thus $\tilde{g}(\boldsymbol{y}) \equiv g(\boldsymbol{R}\boldsymbol{y})$ and $\tilde{f}(\boldsymbol{y}) \equiv f(\boldsymbol{R}\boldsymbol{y})$.

Let
$$\tilde{A}_k \equiv S^{-1} \cdot A_k \cdot \boldsymbol{R}, \tag{26}$$

where S is the $m \times m$ diagonal matrix containing s^j, $j = 1...m$. This leads to the following modified equations[3]

$$\tilde{G}_k(\boldsymbol{y}) = \frac{1}{2}\left\|\tilde{A}_k(\boldsymbol{y} - \boldsymbol{y}_k) - S^{-1}\delta z_k\right\|^2$$
$$= \frac{1}{2}\left\|\tilde{A}_k(\boldsymbol{y} - \boldsymbol{y}_k)\right\|^2 - \langle \tilde{A}_0^T S^{-1}\delta z_k, \boldsymbol{y} - \boldsymbol{y}_k\rangle + \frac{1}{2}\left\|S^{-1}\delta z_k\right\|^2 \tag{27}$$

where
$$\delta z_k = \tilde{f}(\boldsymbol{y}_k) - \hat{z}. \tag{28}$$

Thus the new problem at iteration k is

$$\min\left\{\tilde{G}_k(\boldsymbol{y}) \;\middle|\; \boldsymbol{R}\boldsymbol{y} \in X\right\}. \tag{29}$$

If $n = m$ and $\tilde{A}_k$ is invertible, then the solution to (29) is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k = \boldsymbol{x}_k - \boldsymbol{R}\tilde{A}_k^{-1}S^{-1}\left[f(\boldsymbol{x}_k) - \hat{z}\right] = \boldsymbol{x}_k - A_k^{-1}\left[f(\boldsymbol{x}_k) - \hat{z}\right]. \tag{30}$$

If $n > m$ and $\tilde{A}_k\tilde{A}_k^T$ is invertible, then a solution to (29) is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k = \boldsymbol{x}_k - \boldsymbol{R}\tilde{A}_k^T\left[\tilde{A}_k\tilde{A}_k^T\right]^{-1}S^{-1}\left[f(\boldsymbol{x}_k) - \hat{z}\right], \tag{31}$$

---

1. For the calculation of machine settings, $s^j = 2 \cdot \min(\mathrm{USL}^j - \mathrm{target}^j, \mathrm{target}^j - \mathrm{LSL}^j)$

2. For the calculation of machine settings, r^i is the range of valid settings for control i, i.e. the maximum valid setting value minus the minimum valid setting value.

3. The notation $\langle \bullet, \bullet \rangle$ represents the scalar product.

and this solution as close as possible to the previous solution $\boldsymbol{x}_k$. If $n < m$ and $\tilde{\boldsymbol{A}}_k^T \tilde{\boldsymbol{A}}_k$ is invertible, then the least square error solution to (29) is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k = \boldsymbol{x}_k - \boldsymbol{R}\left[\tilde{\boldsymbol{A}}_k^T \tilde{\boldsymbol{A}}_k\right]^{-1} \tilde{\boldsymbol{A}}_k^T \boldsymbol{S}^{-1} \left[f(\boldsymbol{x}_k) - \hat{\boldsymbol{z}}\right]. \tag{32}$$

Because the above algorithm uses approximations to $f(\bullet)$, a more robust algorithm uses the above equations to calculate a search direction $\boldsymbol{h}_k$ (in equations (30) through (32) substitute $\boldsymbol{x}_{k+1}$ by $\boldsymbol{h}_k$), and then uses an Armijo step size algorithm [20] to determine the actual $\Delta \boldsymbol{x}_k = \lambda_k \boldsymbol{h}_k$, where

$$\lambda_k = \max \{\beta^l \mid \tilde{F}(\boldsymbol{R}^{-1}[\boldsymbol{x}_k + \beta^l \boldsymbol{h}_k]) - \tilde{F}(\boldsymbol{R}^{-1}\boldsymbol{x}_k) \le \alpha \beta^l \langle \boldsymbol{R}^{-1}\boldsymbol{h}_k, \nabla \tilde{F}(\boldsymbol{R}^{-1}\boldsymbol{x}_k)\rangle, \quad l \in N\}, \tag{33}$$

with $\nabla \tilde{F}(\boldsymbol{R}^{-1}\boldsymbol{x}_k) = \tilde{\boldsymbol{A}}_k^T \boldsymbol{S}^{-1} \left[f(\boldsymbol{x}_k) - \hat{\boldsymbol{z}}\right]$, $\alpha \in (0, \frac{1}{2})$, $\beta \in (0, 1)$, and

$$\tilde{F}(\boldsymbol{y}) \equiv \frac{1}{2} \sum_{j=1}^{m} \left\{ \frac{1}{s^j} \left[f^j(\boldsymbol{R}\boldsymbol{y}) - z^j\right] \right\}^2, \tag{34}$$

so that

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k = \boldsymbol{x}_k + \lambda_k \boldsymbol{h}_k. \tag{35}$$

Whenever there are problems with performing the matrix inversion in (30), (31), or (32), a steepest descent method can be used to find the search direction[1]:

$$\boldsymbol{h}_k = \boldsymbol{x}_k - \boldsymbol{R}\tilde{\boldsymbol{A}}_k^T \boldsymbol{S}^{-1} \left[f(\boldsymbol{x}_k) - \hat{\boldsymbol{z}}\right]. \tag{36}$$

For most systems, the set of valid input recipes X is constrained. Consider a set of constraints

$$x_{\min,i} \le x_i \le x_{\max,i}, \qquad i = 1 \ldots n \tag{37}$$

1. A more sophisticated method of dealing with such inversion problems is to perform a singular value decomposition and eliminate the dimensions corresponding to near zero singular values to obtain a transformation into a space with fewer dimensions, so that the required matrix inversion can be accomplished. This analysis is beyond the scope of this report.

where $x_{\min,i}$ is the minimum valid value for setting i, and $x_{\max,i}$ is the maximum valid value for setting i. Whenever a recipe is calculated, the algorithm must be able to remain within the constraints given by (37). To handle these constraints, a feasible modification[1] to the above algorithms freezes any constraint violating input value (at the minimum or maximum according to the nature of the violation), and reduces the dimension of the input space by one. Thus the optimization can continue with the remaining inputs. For a more robust system, a more general constrained optimization algorithm could be applied [20].

1. The reliability of this modification depends on the assumption that all of the modeled output values are monotonically either increasing or decreasing with respect to each individual setting value, an assumption which is satisfied by most semiconductor processing equipment.

2.5 Application of the Model Update and Recipe Update Algorithms

The presented methods for checking and updating equipment models and for calculating a new equipment setting recipe are built into the BCAM control applications. The model update algorithm is used as part of the feedback system to maintain current models for processing equipment. The recipe update algorithm is used for several purposes; it is used for initial setting calculations, for recipe updates whenever current equipment models change, and for feed-forward calculation of workcell controls.

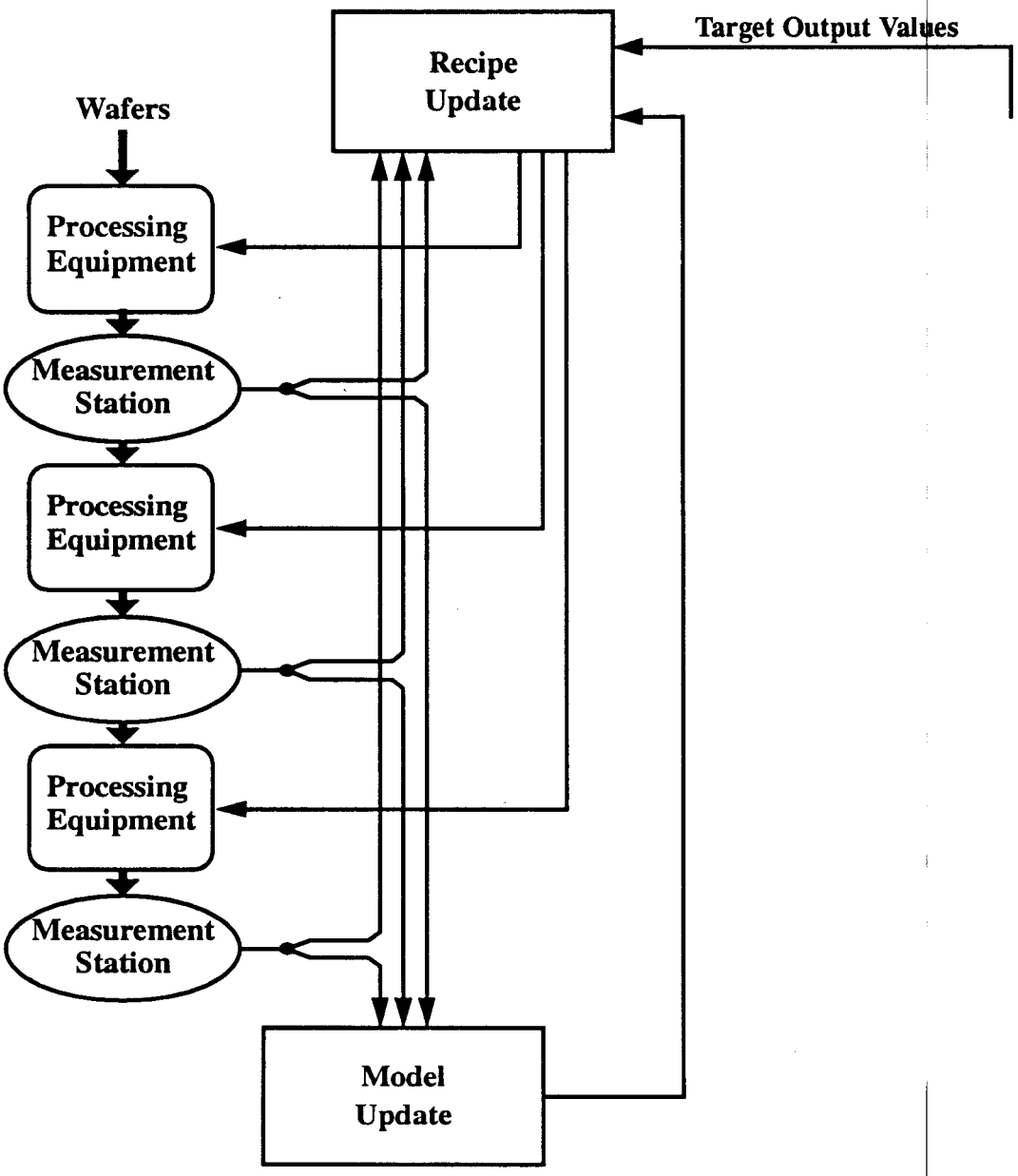

Figure 1 Application of Model and Recipe Update Algorithms

Chapter 3 BCAM Implementation

3.1 Introduction

Because the accurate control of manufacturing processes is critical to the production of integrated circuits, a control scheme is needed so that deviations from product specifications may be compensated by automatic adjustments to the process. This control must exploit the interdependence of the various steps involved in production. Software has been developed to utilize equipment models for supervisory control. The software uses the models for process simulation and recipe generation. This recipe generation is used to accommodate product specifications and to implement feed-forward control; in order to compensate for deviations in the middle of a process run, feed-forward control adjusts the settings of subsequent process steps.

Feedback control is initiated by a model-based control and monitoring scheme. Alarms using statistical analyses [5] are employed to detect consistent departures in equipment performance from the performance predicted by the equipment models. Once this departure has been established, the model is modified through a model update algorithm [7] and the equipment control settings are correspondingly adjusted. The workcell controller has several process analysis functions which examine equipment interconnection information to determine the order of processing steps, to predict performance of an entire workcell, and to perform sensitivity analyses on the workcell. Feed-forward control is initiated whenever the projected properties of a wafer lot fall outside of specifications. It is implemented by adjusting the recipes (control settings) of subsequent processing steps. If projections predict that feed forward control cannot bring the lot back within the acceptance limits [13], the lot will be discarded or re-processed.

This chapter gives an overview of the implementation of the BCAM Environment. Low level implementations details are described in the BCAM Environment Programming Manual presented in Chapter 5.

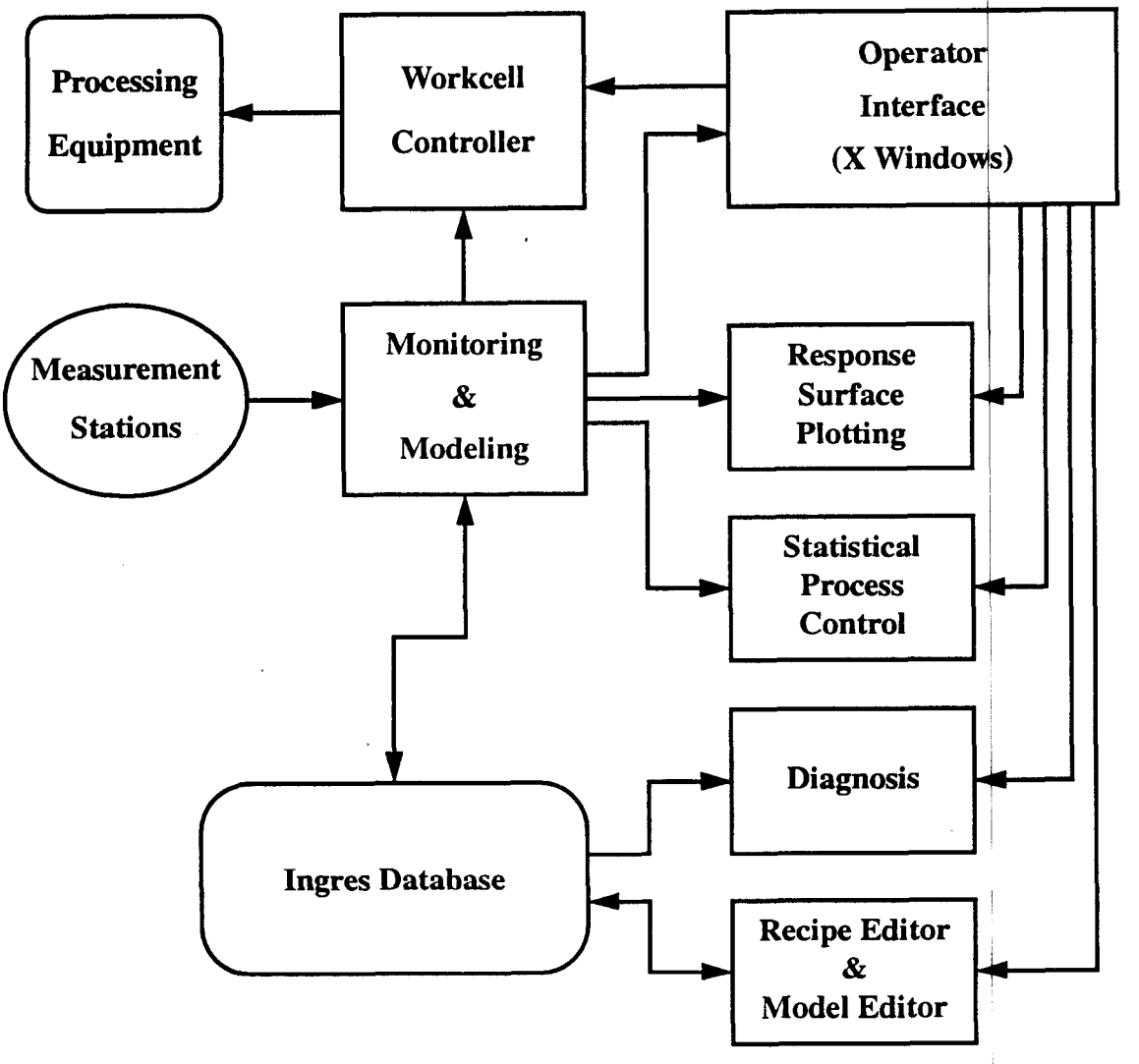

Figure 2 The BCAM Environment

3.2 The Ingres Database

The BCAM environment must operate in a multiple user system with security precautions and the ability to avoid conflicts when more than one user accesses the database simultaneously. The environment therefore makes use of the Ingres database program. This program allows multiple users to interact with a central database. Ingres

also allows different levels of permissions, such as those of an operator or an engineer. Ownership and time stamps are maintained for all tables stored in the database, thereby preserving security and preventing access conflicts. The BCAM Environment uses Ingres library functions to store and retrieve equipment recipes and models in the database. In order to enhance the portability of the BCAM Environment, the code has been developed to allow easy modification for use with other database programs.

3.3 The Role of X Windows in the BCAM Environment

The X window system provides a means to implement an aesthetically pleasing, user friendly interface to the BCAM environment. X windows were chosen because they are available throughout the industry and are standardized to work on most workstations. A commercially available graphical interface package was also investigated, but it was determined to be inappropriate for the BCAM Environment. This study is described in Appendix J.

3.4 Operations on Individual Machines

The most basic function of the BCAM environment is the selective activation of individual machines. Once activated, the BCAM system can interact with each machine in a variety of ways.

By accessing the system database, a recipe editor[1] allows machine settings to be retrieved, edited, and saved as desired. Ownership and time stamps are maintained on all recipes. Recipes may also be calculated to meet specified targets.

Analytical models for the machine are also loaded from the database. The implementation allows full interaction with these models, including editing, automatic model updates, response surface viewing, and storage of all models in the database. As

1. The recipe editor has been developed by Hao-Cheng Liu of the Berkeley Computer-Aided Manufacturing group.

models are updated, corrections are also stored in the database[1]. Ownership and time stamps are maintained on all models. The operator may use these models for prediction and recipe generation.

Several types of equipment connection information may also be addressed by the operator. Inputs which must remain constant or are uncontrollable can be designated as *uncontrollable*. Outputs which are considered important, or for which specifications are given, can be designated as *final outputs*. Often the output of one machine affects the performance of the following processing step. Therefore, the operator can connect the output of any machine to the input of any other machine. The BCAM environment checks for validity of such connections by comparing the units of the addressed input/output pair.

3.5 Process Analysis Functions

3.5.1 Deduction of Process Order

After the user has designated the equipment interconnections, the workcell controller must deduce the order in which the machines will operate. The controller does this by following connection paths to determine which machine depends on the greatest hierarchy of other machines for its input values. This machine is placed at the end of the processing line. The controller then checks for the machine with the next greatest dependence and places it second to last. The method is continued until all connected machines have been included. This method can handle any configuration of machines and will automatically detect misconfigurations which lead to loops. Once the workcell configuration has been determined, the controller may consider the whole workcell (or parts of it) as a single operation.

1. Automatic storage of model corrections has not yet been implemented.

3.5.2 Workcell Performance Prediction

For several applications, the workcell controller must be able to predict the final outputs of a workcell based on the individual controls applied to all machines in the workcell. This is accomplished by first predicting the outputs of the first machine, feeding these outputs to the inputs of the following machine, and continuing until the predictions for the last machine have been calculated, thus yielding the desired result.

3.5.3 Workcell Sensitivity Analyses

In order to determine the sensitivity of any workcell final output to any workcell input, the controller uses a recursive algorithm. This algorithm uses the workcell's interconnection information to implement a multidimensional evaluation of differentiation of composed functions. The level of hierarchy for the composition (and thus the level of recursion) depends on the number of machines between the machine receiving the input and the machine whose output is considered.

3.6 The Workcell Operations

In addition to providing control over a processing line, the controller must be able to conduct simulations of the process. This will allow operators to perform preliminary tests of new process configurations or control schemes without expensive and time consuming wafer processing.

The run-by-run control system under development by the BCAM group uses models for the individual machines in a process to build a model of the entire process. A process specifications menu enables the user to set the product's desired characteristics, and then the controller will automatically calculate the optimum equipment settings to meet that product's specifications. These settings make up the process recipe. Once this has been accomplished, the controller may begin processing wafers.

As wafers are processed, the BCAM Environment makes a record of each machine's performance. This record is used by the alarm generation module, the model update algorithm, and the graph generation module.

3.7 The Interface to Other BCAM Applications

The Berkeley Computer-Aided Manufacturing research group has developed several applications for manufacturing. These applications can be invoked for use with a piece of equipment through the BCAM environment.

Formal alarm generation algorithms have been developed by Sovarong Leang [22]. They will be integrated into the monitoring systems of the BCAM environment.

Real time statistical process control (SPC) algorithms have been implemented by Eddie Wen [22][23]. This implementation includes graphical displays. The SPC application functions for measurements taken from the Berkeley Microfabrication Laboratory's Lam Autoetch 490.

An application which performs diagnoses on equipment malfunctions has been developed by Dr. Gary May, Hao-Cheng Liu, and Sherry Lee [16][17]. This diagnosis module is available for the Berkeley Microfabrication Laboratory's Lam Autoetcher through the BCAM environment.

An application to create two and three dimensional response surface plots of equipment models has been developed by Eddie Wen and Eric Braun for use with the BCAM environment[1] [22].

1. This model plotting algorithm is currently in the process of integration into the BCAM environment.

Chapter 4 BCAM Environment User's Manual

4.1 Introduction: Starting the BCAM System[1]

To enter into the BCAM environment, first change directories to "~bcamdev/bin" then enter the command "source ~bcam/src/main/.bcam" and then enter the command "./BCAM"

An introductory widow will appear. Click on "Ok" to continue. At this point the main menu will appear. To activate some equipment, click on the "EQUIP" selection and a menu of available equipment will appear. Clicking on an equipment name activates that equipment, and a window for the equipment will appear. Several machines may be activated at any given time.

The individual equipment windows allow a multitude of operations. Equipment setting recipes may be edited, stored, and retrieved from the BCAM database. Similarly, equipment performance models may be edited, stored, and retrieved. The equipment window also facilitates the definition of equipment interconnections for use with the BCAM workcell controller application. In addition to the workcell controller, the BCAM environment supports Diagnosis and Statistical Process Control (SPC) applications; these applications are also activated from the equipment window.

1. These instructions are for the current experimental implementation on U. C. Berkeley's radon Sun4 computer.

4.2 The Equipment Window

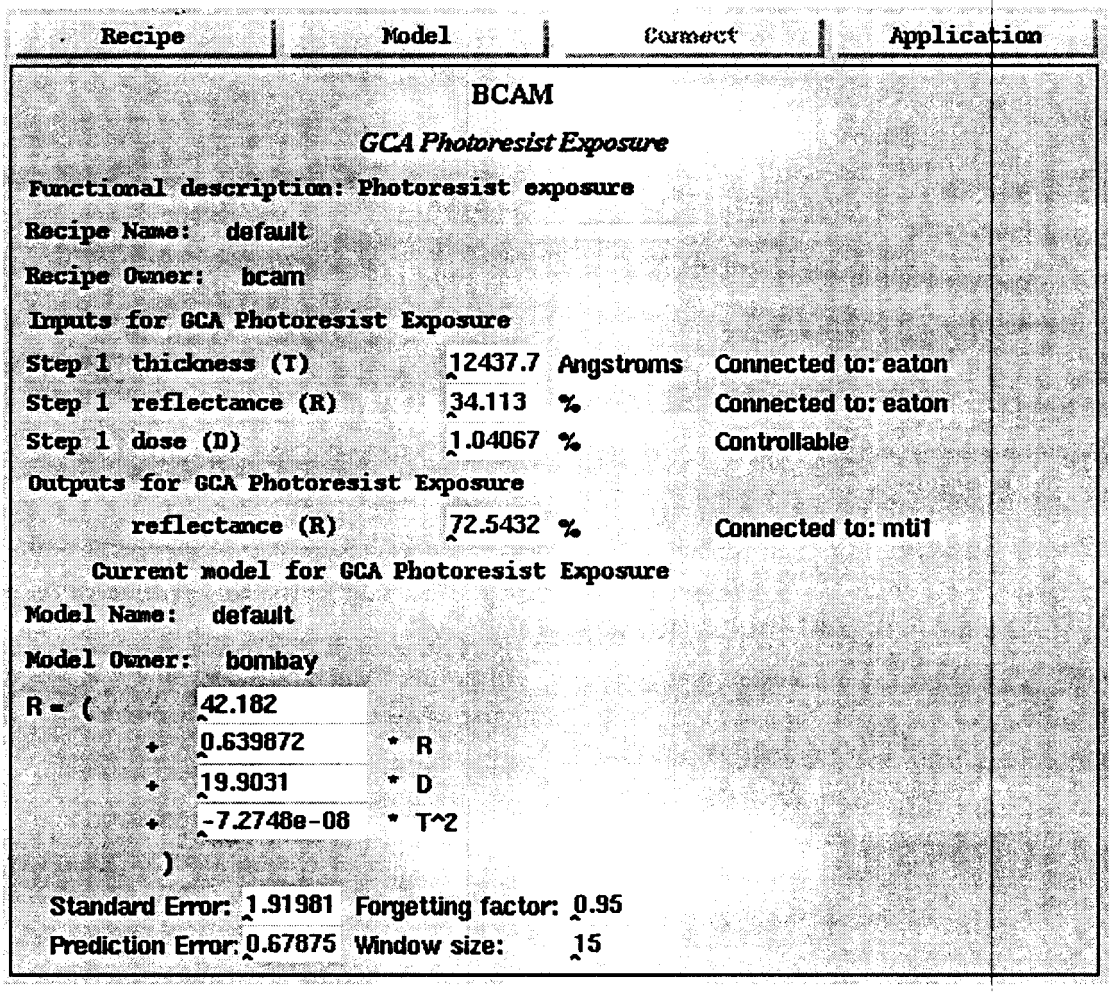
Recipe
Model
Connect
Application
BCAM
GCA Photoresist Exposure
Functional description: Photoresist exposure
Recipe Name: default
Recipe Owner: bcam
Inputs for GCA Photoresist Exposure
Step 1 thickness (T) 12437.7 Angstroms Connected to: eaton
Step 1 reflectance (R) 34.113 % Connected to: eaton
Step 1 dose (D) 1.04067 % Controllable
Outputs for GCA Photoresist Exposure
reflectance (R) 72.5432 % Connected to: mti1
Current model for GCA Photoresist Exposure
Model Name: default
Model Owner: bombay
R = (42.162
+ 0.639872 * R
+ 19.9031 * D
+ -7.2748e-08 * T^2
)
Standard Error: 1.91981 Forgetting factor: 0.95
Prediction Error: 0.67875 Window size: 15

Figure 3 The GCA Stepper Equipment Window

4.2.1 General Description

Most operations of the equipment window are accessed through the menu bar at the top of the window. This menu bar has four components: Recipe, Model, Connect, and Application. In addition, values for inputs, outputs, and model coefficients may be directly edited in the equipment window, and connection information may be designated by clicking on the names of the inputs and outputs.

4.2.2 The Equipment Recipe Menu

The Recipe Menu is used to access all functions associated with the currently stored recipe of control and input values for a machine. The machine is also deactivated through this window.

- Load Recipe...—Select and load a recipe from the BCAM database.
- Load Default Recipe—Load the user's default recipe from the BCAM database; if the user has no default recipe, the BCAM system default is loaded.
- Save Recipe—Save the current recipe into the BCAM database under the current name.
- Save Recipe As...—Specify a name for the current recipe and save it into the BCAM database.
- Restore Recipe—Restore the recipe to the values it had at the last operation.
- Delete Recipe—Delete a recipe from the database.
- Output Targets—Set target values and tolerances for all final outputs of this machine.
- Output Specifications—Set target values and specification limits for all final outputs of this machine.
- Calculate Recipe—Generate a recipe to produce the final output values specified by targets.
- Download Recipe—Download a recipe of controls to the physical equipment.
- Deactivate this Machine—Eliminate this equipment window and deactivate the corresponding machine.

4.2.3 The Equipment Model Menu

The Model Menu is used to access all functions which deal with equipment models.

- Load Model...—Select and load a model from the BCAM database.
- Load Default Model—Load the user's default model from the BCAM database; if the user has no default model, the BCAM system default is used.
- Save Model—Save the current model into the BCAM database under the current name.
- Save Model As...—Select a name for the current model and save it into the BCAM database.
- Display Original Model—Display the model as it existed before any feedback corrections.
- Display Current Model—Display the current adaptive equipment model (original model plus correction model).
- Confirm Current Model—Confirm the currently edited and displayed model and take it as the current adaptive model.
- Predict—Predict the outputs generated by the current input recipe.

- Check Model—Using equipment performance data, execute statistical regressions to check (and if needed, correct) the current adaptive model.
- Response Surface Plots—Activate the response surface plotting facility for equipment models.
- Load Simulator Model...—Select and load a simulator model from the BCAM database.
- Load Default Simulator Model—Load the simulator with the user's default model from the BCAM database; if the user has no default model, the BCAM system default is used.
- Display Simulator Model—Display the model currently being used by the BCAM equipment simulator.
- Confirm Simulator Model—Confirm the currently edited and displayed model and take it as the current simulator model.

Note that there are two distinct models used in the BCAM environment. The first is a the current (controller) model. This model is used for all prediction and control purposes, including the feedback and feed-forward control algorithms. The original version of this controller model is also maintained and may be viewed with the Display Original Model command. The Check Model command is used to manually initiate a check and update of this model. The other model is the simulator model; this model is used purely for demonstration purposes. Unless the simulator is explicitly specified, all references to a model are to the controller model.

4.2.4 The Equipment Window—Defining Equipment Connections

To connect the input (setting, control, or incoming measurement) of one machine to the output (usually a post-processing measurement) of another machine, click successively upon the appropriate input name and output name. The display will then change to describe this connection. To remove the connection, click upon the name of the input side of the connection (i.e. where that parameter shows up as an input).

For control purposes, not all outputs are critical to the final product of that workcell. To designate an output value to be of final importance, double click upon the name of that output. The window will then show that output to be a "final output." The operator is

automatically prompted for specifications whenever a final output is designated. Another two clicks upon the output name return the output to normal status. Multiple final outputs may be designated.

In many cases, input parameters which affect machine performance may not be controllable. Furthermore, it may sometimes be desirable to hold specific machine control inputs constant. In such cases, the relevant inputs may be designated as uncontrollable. To declare an input uncontrollable, double click upon the name of the input. The display will then reflect this designation. Another single click reverses the designation.

Because the workcell configuration depends on equipment interconnections, connection information may not be changed while the BCAM Workcell Controller is active.

4.2.5 Example: The Interconnection of a Photolithography Workcell

The following figure shows a photolithography workcell using three pieces of equipment. Two measurements are taken on the wafers exiting the photoresist spinner: photoresist thickness (T) and reflectance (R). The photoresist reflectance is again measured after exposure (the thickness is not affected by the exposure). Finally, the critical dimension (CD) is measured after the development. Since the thickness and reflectance measurements are considered inputs to the following processing steps (they appear in model equations used by the BCAM controller), they are connected as depicted

in the figure.

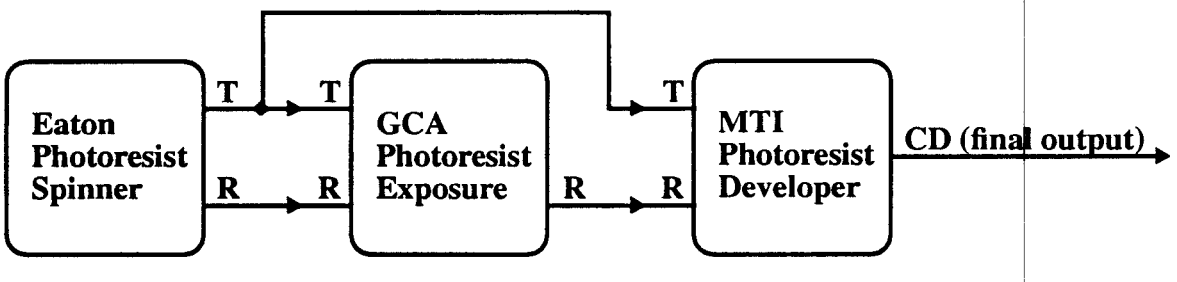

Figure 4 Photolithography Workcell Interconnection

Once the three machines have been activated, the following sequence of mouse clicks will set up the connections in the BCAM environment:

- Double click on the developer's output CD to designate the final output.
- Click on the developer's input thickness, and click on the spinner's output thickness to connect.
- Click on the exposure's input thickness, and click on the spinner's output thickness to connect.
- Click on the developer's input reflectance, and click on the exposure's output reflectance to connect.
- Click on the exposure's input reflectance, and click on the spinner's output reflectance to connect.

The workcell connection information is thus completed. The workcell controller may now be invoked through the Equipment Applications Menu described in 4.2.7.

4.2.6 The Equipment Connections Menu

- Connect Input—Change the status of an input or connect the input to the output of another machine.
- Connect Output—Change the status of an output or connect the output to the input of another machine.

Note that connection information may also be specified by clicking directly on input and output names (4.2.4). This more direct method is generally preferred.

4.2.7 The Equipment Applications Menu

This menu is used to access other major BCAM applications.

- Workcell Controller—Activate the BCAM Workcell Controller using current equipment interconnection information. (see below 'The Workcell Window')
- Statistical Process Control—Activate the BCAM Real-Time Statistical Process Control monitoring system [22][23]. This application is not available for all machines.
- Diagnosis—Activate the BCAM Diagnosis module [16][17]. This application is not available for all machines.
- Response Surface Plots—Activate the response surface plotting facility for equipment models [22].

4.2.8 The Equipment Window—Editing Values

The values shown for input values, output values, and model coefficient values are editable. To edit these values, click the mouse pointer upon the value to be edited. The value may then be changed using the usual editing keys. All commands from the equipment window check these editing fields for new values before executing. The workcell controller, however, does not automatically check model coefficient values, and it is therefore desirable to manually confirm these values (from the Model pulldown menu) after editing. This is necessary, for example, whenever the simulator model is changed.

4.3 The Workcell Window

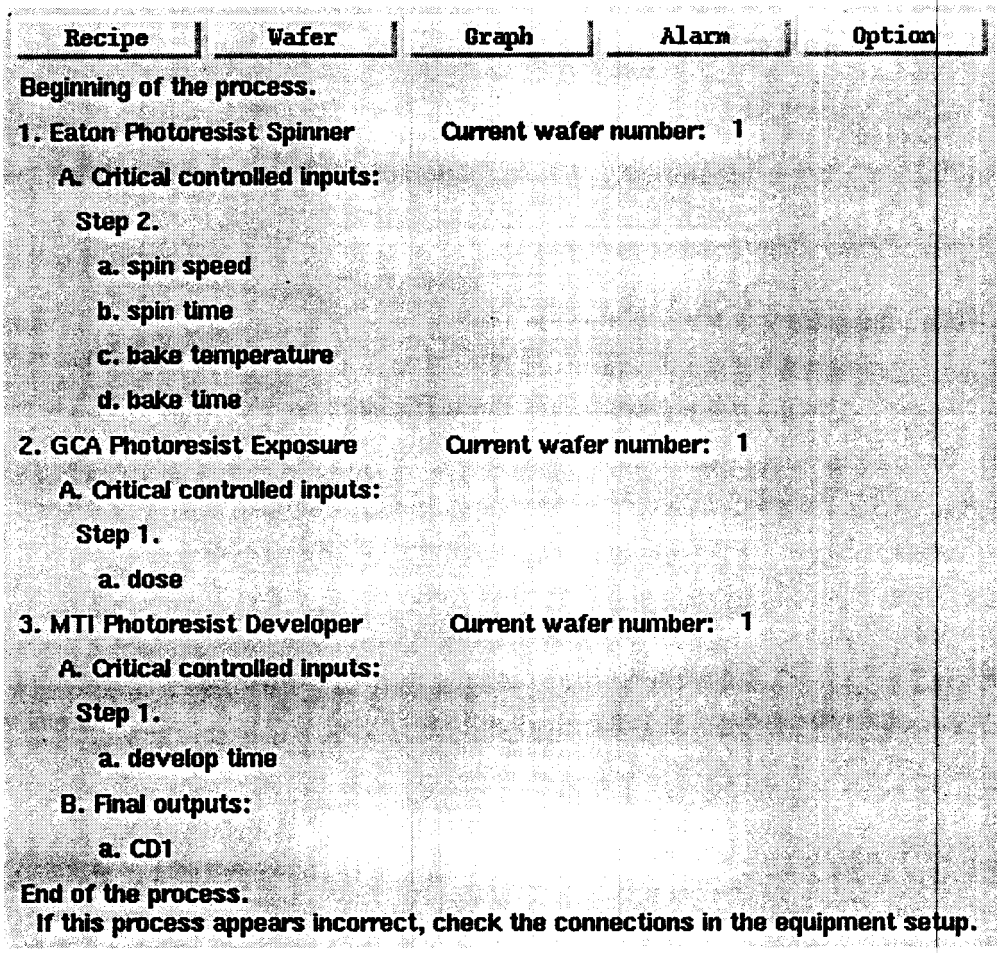

Figure 5 The Photolithography Workcell Window

The Workcell Window is used to manipulate a group of machines connected into a processing workcell. This window is activated from the Equipment Window's Applications Menu by choosing the Workcell Controller. Equipment interconnections must be defined before invoking the workcell window, and they cannot be changed while the workcell window is present. To disengage the workcell window, use the Exit Process Workcell command from the Workcell Recipe Menu.

4.3.1 The Workcell Recipe Menu

This menu accesses functions for manipulating workcell recipes. The Workcell Window is also disengaged through this menu. In addition, this menu can be used to specify target values for the final outputs of the workcell. Upon receipt of the specifications, the controller automatically attempts to calculate a recipe of controls which will yield the desired results.

- Edit—Edit the current workcell controls.
- Load...—Load a new recipe for the workcell controls from the database.[1]
- Load Default—Load default controls for the workcell
- Save—Save current controls into the database.
- Save As...—Save current controls under a specified name.
- Final Output Targets—Specify final output targets and tolerances.
- Final Output Specifications—Specify final output targets and both upper and lower specification limits.
- Calculate Recipe—Calculate a recipe to meet output specifications.
- Exit Processing Workcell—Exit the workcell mode and Workcell Window.

4.3.2 The Workcell Wafer Menu

This menu is used to initiate actions upon or about wafers, including actual operation of the workcell.

- Run Wafer Through Workcell—Send a wafer to the workcell for processing.
- Simulate a Wafer Run—Simulate the workcell's operation on a single wafer (used for demonstration).
- Simulate N Wafer Runs...—Simulate a series of wafers through the workcell (used for demonstration).
- Predict Final Outputs—Predict the final outputs of the workcell using the current recipe.
- Sensitivities—Display the sensitivities of the final outputs to the workcell's controls.[2]
- Evaluate Cpk Capability—Evaluate the C_{pk} capability of the workcell.[3]

---

1. Database operations for the workcell have not yet been implemented. These operations are, however, available from the individual equipment windows.
2. Sensitivity displays have not yet been implemented.
3. C_{pk} evaluation has not yet been implemented.

4.3.3 The Workcell Graph Menu

This menu accesses the graphical utilities associated with workcell operation.

- Outputs—Graph the recorded output measurements of all workcell machines.
- Final Outputs—Graph the final output measurements of the workcell.
- Prediction Errors—Create regression control chart on prediction errors.
- Autocorrelation—Graph the autocorrelations of prediction errors.
- CUSUM—Show a CUSUM graph of final output measurements.
- Inputs—Graph the controls used versus wafer numbers.
- Normalized Inputs—Graph all inputs normalized to fit on a single graph.
- Adaptive Models—Graph the coefficients of the adaptive models.

4.3.4 The Workcell Alarm Menu

This menu is used to access BCAM alarm utilities. Manual alarms can be issued using this menu[1].

- Invalidate History—Specify that all previously recorded measurements should be ignored by the model update algorithms.
- Discard History—Purge all recorded measurements from memory.

4.3.5 The Workcell Options Menu

The workcell controller allows the user to specify several optional parameters for dealing with the workcell. These parameters are accessed through the Workcell Options Menu.

- Turn On/Off Feed-Forward Control
- Turn On/Off Feedback Control
- Turn On/Off EVOP[2]

1. This menu will be expanded in the future [22].
2. Evolutionary operation has not yet been implemented.

Chapter 5 BCAM Environment Programming Manual

5.1 Introduction: The Basic Program Structure

To aid in the ongoing quest to improve integrated circuit manufacturing techniques, the BCAM group has developed an object-oriented software library describing the fabrication equipment. The library provides a set of functions for interacting with equipment and databases and for manipulating various models of machine performance. Once the BCAM software has loaded equipment information into memory, it can perform predictions, sensitivity studies, simulations, recipe generation and statistical process control. Also included in the software are procedures for updating machine performance models, alarm generation, and diagnostic analyses. The BCAM software is organized as a library which can be accessed by all modules in the BCAM architecture (e.g. workcell controller, recipe editor, malfunction diagnosis, and statistical process control). The entire library is written in C++, an object-oriented superset of the C programming language. C++ extends the normal capabilities of C by adding features such as data abstraction, message-passing, polymorphism, inheritance, and class hierarchy [11].

The BCAM environment is built around a C++ class called *machineClass*. This data structure includes all the information necessary to interact with the physical machine. Member functions of this class or its member classes are used for BCAM actions related to the individual machine. For every equipment activated, a separate instance of a *machineClass* structure is created. When the BCAM Environment is active, each *machineClass* structure manifests itself on the screen in an *Equipment Window*.

Although the greatest care has been taken to keep this information current, herein may be discrepancies with the current code. For the most recent information, please refer to the source code.

5.2 The Creation of the Main Windows

The program first creates an entrance window which introduces the operator to the system. When the operator is ready to begin, a mouse click will bring up the main menu for the environment. This main menu is created using the function *Widget MakeMainMenu()*. Callbacks are set up from this menu to access the highest level operations, most importantly the activation of equipment. The functions which handle these actions are located primarily in the files 'controllerEntrance.cc', 'equipSetup.cc', and 'mainMenu.cc'.

Whenever a machine is activated, an instance of *machineClass* is created and information from the database is loaded into this structure. Then a call to *Widget machineClass::MakeEquipWindow()* creates the actual equipment window. Integral in the creation of the equipment window is the creation of the bar of pulldown menus from which most operations are accessed. This is accomplished through use of the functions *Widget MakePulldownBar()* and *Widget MakePulldownMenu()*. The functions which handle these actions are located primarily in the files 'equipWindow.cc' and 'mainMenu.cc'.

The workcell controller window is created in a similar fashion to the equipment window. The functions which handle these actions are located primarily in the files 'process.cc' and 'mainMenu.cc'.

5.3 Naming Conventions

Except in cases of imported code, the following naming conventions have been adhered to:

- Macro names are all capital letters (e.g. MACRO).
- Function names are lower case with all individual words beginning with capital letters for ease of reading (e.g. FunctionName).

- Variable names (except for those of Widget type) are lower case with imbedded words beginning with capital letters for ease of reading (e.g. variableName).
- Variable names of type Widget are lower case with imbedded words separated by the underscore character (e.g. widget_variable_name).
- Type and class names follow the same conventions as variable names.

5.4 Source Code File Hierarchy

The source code file hierarchy (from top down) is approximately organized in the following order:

- BCAM.cc controllerEntrance.cc
- mainMenu.cc
- equipSetup.cc
- machineRW.cc
- modelRW.cc
- modelIngres.scc recipeIngres.scc
- equipWindow.cc functions.cc messageWidget.cc makeWarning.cc
- connections2.cc connections.cc modelChoice.cc
- process.cc interfaces.cc
- generateRecipe.cc
- modelUpdate.cc
- graphsBCAM.cc graphRM.cc
- diagnosis.cc
- modelEval.cc
- steps.cc
- modelSpecFn.cc
- mymath.cc

5.5 Structure Declarations

Descriptions of most data classes and their member functions are contained in the declarations file '*machineTypes.h*'. The principal classes have the following hierarchy[1]:

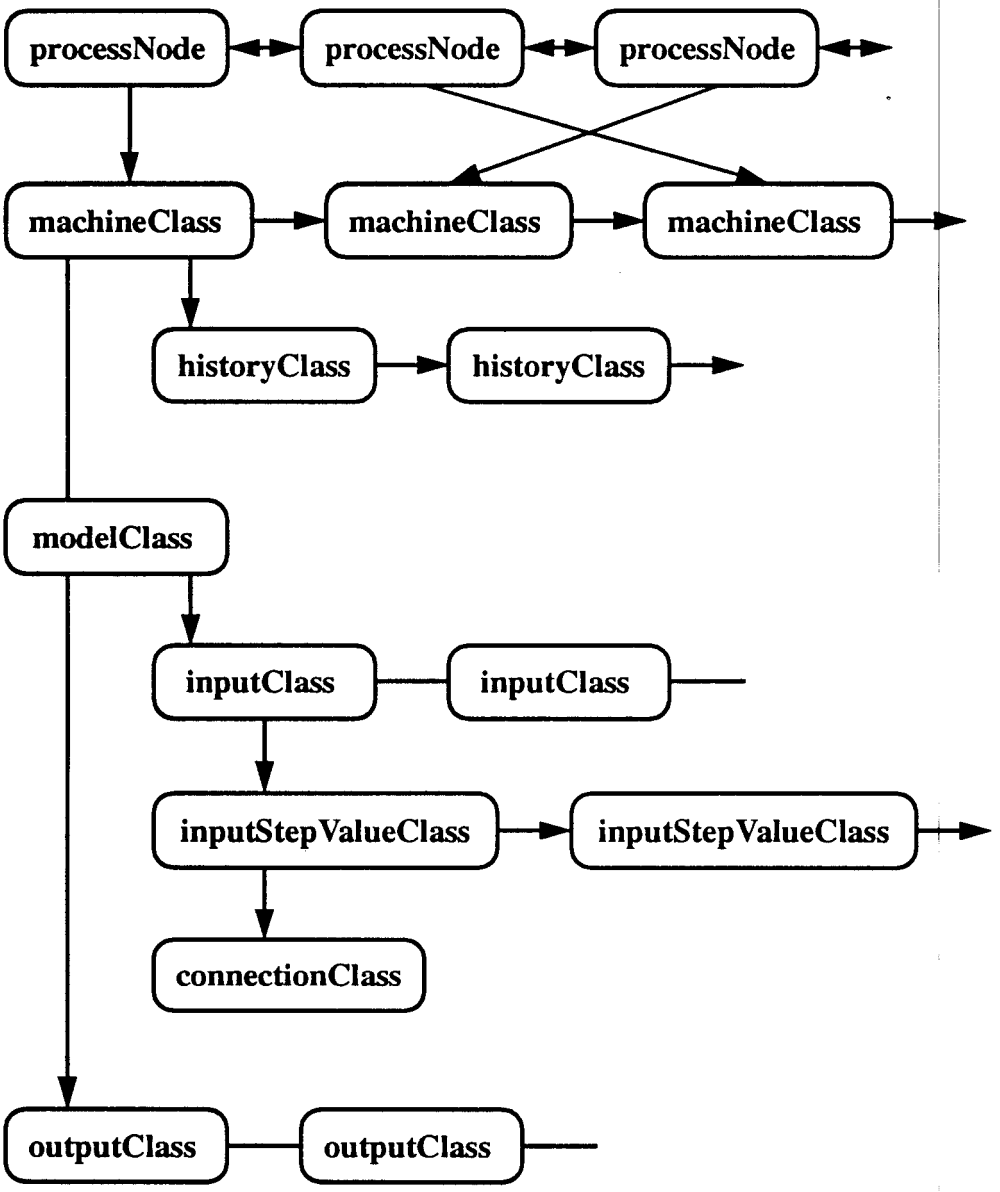

Figure 6 BCAM C++ Data Structures

The principal members of some of the important structures are as follows:

1. Arrows indicate pointers (horizontal arrows form linked lists); vertical bars indicate membership; and horizontal bars indicate consecutive members of an array.

5.5.1 class machineClass

The core of the BCAM environment is the class machineClass. It has many member data structures and functions, only the most important of which are described here. Some of these members are themselves classes, and these classes are also described in this document.

- *Widget window* — The equipment window displayed on the screen.
- *modelClass* model* — The current models for machine performance. Included are the models both for control purposes and for simulation.
- *historyClass* history* — A linked list of historical equipment performance data.
- *int waferNum* — The identification number of the wafer currently being processed by this machine.
- *int GetInputIndex(char* inputName)* — Return the array index (into *model.inputs*) of the input named *inputName*.
- *int GetOutputIndex(char* outputName)* — Return the array index (into *model.outputs*) of the output named *outputName*.
- *double predict(int outputNum)* — Evaluate and return the value predicted for output number *outputNum* using the controller machine performance model and the current recipe for inputs.
- *void simulate()* — Simulate all outputs of this machine including Gaussian noise. Simulated output values are stored in the *model.outputs* array.
- *int ModelUpdate()* — Look at the historical performance data of this machine and determine if an update to the machine's controller model is statistically justified. If a correction is justified, it is made, and the function returns a positive value. Otherwise the function returns zero. A negative return value indicates an error.
- *machineClass* next* — A pointer to the next machine in the linked list of machines.

5.5.2 class modelClass

This class contains most of the information necessary for modeling or simulating a machine's performance.

- *char* equipName* The abbreviated name of the machine.
- *char* formalName* The formal name of the machine
- *char* user* The login name of the current BCAM user.
- *double** exp_mat* A matrix containing the exponents for the inputs in all the terms used in the equipment model. Each row corresponds to one term. Column indices correspond to input numbers. This member is private.
- *int** spec_fn* A matrix containing special function codes for the inputs in all the terms used in the equipment model. Each row corresponds to one term. Column indices correspond to input numbers. The code zero indicates that no special function is to be used. Special functions are applied to inputs before exponents. This member is private.
- *inputClass inputs[MAX_INPUTS]* The array of input descriptors of the machine. The current recipe is located within.
- *int input_cnt* The number of inputs for the machine.
- *outputClass outputs[MAX_OUTPUTS]* The array of output descriptors of the machine.
- *int output_cnt* The number of outputs for the machine.
- *double simulate(int outputNum)* Simulate and return the value for output number *outputNum* using the simulator performance model and the current recipe for inputs.
- *double predict(int outputNum)* Predict and return the value of output number *outputNum* using the controller's current adaptive model and the current recipe for inputs.

5.5.3 class processNode

Another important class is the class *processNode*. This class is used to describe a workcell containing several machines. This class is used extensively by the BCAM Workcell Controller. When the workcell controller is invoked, the machine interconnections are analyzed and a linked list of *processNode*s are created to describe the order in which the machines operate, i.e. describe the process. Each process node has the following members:

- *machineClass* machine*
  A pointer to the *machineClass* structure corresponding to the machine at this node in the process workcell.
- *processNode* preceding*
  A pointer to the preceding process node in the workcell.
- *processNode* following*
  A pointer to the following process node in the workcell.
- *processNode* start* A pointer to the first process node in the workcell.
- *processNode* end* A pointer to the last process node in the workcell.
- *void CalcControls()* Calculate the values for the controllable inputs for this machine and all following machines in order to reach target output values. Calculated values are automatically stored in the *inputs* member.

5.5.4 class inputClass

The class *inputClass* contains all the information about an input for a machine and functions for manipulating that input. Descriptions of some members follow:

- *char* name* The name of the input.
- *char* abbr* An abbreviation of the name of the input.
- *char* units* The unit of measure associated with values of this input.
- *double min* The minimum recommended value for this input.
- *double max* The maximum recommended value for this input.
- *inputStepValueClass* valueList*
  A list of values that this input takes at each step in the operation of this machine. This is a private member.
- *double Value(int step)* Return the value of this input at step *step*.
- *double SetValue(int step, double newValue)*
  Assign a new value for this input at step *step*. Also return this new value.
- *boolean Connected()* Return TRUE if this input is connected to another machine. Otherwise return FALSE.

5.5.5 class inputStepValueClass

This class is used to form a list which stores the current recipe values of an input at all relevant steps in the equipment operation. Connection information is also stored here.

- *double value* The value of an input at this step. This is a private member.
- *int step* The step number for this step.

- *boolean critical* — True if this is a critical step for a machine's operation. False otherwise. Only one step should be critical for any input. This information is used by the model evaluation routines.
- *connectionClass connection* — Connection information for this input.
- *inputStepValueClass* next* — The next step in this linked list.

5.5.6 class outputClass

The class *outputClass* contains all the information about an output for a machine and functions for manipulating the outputs. Descriptions of some members follow:

- *char* name* — The name of this output.
- *char* abbr* — An abbreviation of the name of this output.
- *char* units* — The unit of measure for the values of this output.
- *double value* — The current value of this output. This is a private member.
- *double min* — The minimum feasible value for this output.
- *double max* — The maximum feasible value for this output.
- *double upperSpecLimit* — The upper specification limit for this output in a particular manufacturing application.
- *double lowerSpecLimit* — The lower specification limit for this output in a particular manufacturing application.
- *coeffsType coeffs* — The original controller model coefficients for this output. This is a private member.
- *coeffsType corr_coeffs* — The controller model correction coefficients for this output. This is a private member.
- *coeffsType simul_coeffs* — The simulator model coefficients for this output. This is a private member.
- *double Value()* — Return the current value of this output.
- *double SetValue(double newValue)* — Assign a new value to this output. Also return this new value.
- *boolean Connected()* — Return TRUE if this output is connected to another machine. Otherwise return FALSE.

5.5.7 class historyClass

An instance of this class stores information about the processing of a particular wafer on a machine. Each machineClass structure includes a linked list of these history structures.

- int waferNum — The wafer identification number
- double* recipe — The controls used to process this wafer on this machine
- double** modelCoeffs — The coefficients that the adaptive model used when this wafer was processed
- double* outputValues — The output values measured for this wafer
- double* upperSpecLimits — The upper specification limits on the output measurements of this wafer
- double* desiredOutputValues — The targeted values for the output measurements of this wafer
- double* lowerSpecLimits — The lower specification limits on the output measurements of this wafer
- double* predictedOutputValues — The output values which were predicted by the adaptive model at the time this wafer was processed
- double* trueOutputValues — In the case where a simulator was used, the true state of the simulator is reflected here by recording noiseless output values of the simulator.
- historyClass* next — A pointer to the next oldest wafer record

Chapter 6 Conclusions & Future Work

6.1 Conclusions

An implementation of an integrated circuit manufacturing control and monitoring environment has been realized using C++ and X windows. This environment is named the Berkeley Computer Aided Manufacturing Environment. The environment provides interfaces to a variety of manufacturing applications, including statistical process control, malfunction diagnosis, and response surface visualization of equipment models. The environment also provides a complete control system for processing workcells. The environment interfaces with an Ingres database for maintenance of machine operation recipes and equipment model records. In order to provide automated equipment control, the controller uses an optimization algorithm for recipe updates and a statistically based algorithm for model updates.

6.2 Database Storage of Wafer Measurements

Currently, the BCAM environment maintains wafer measurement records only in memory while running. These measurement records, along with the machine controls used to process the wafers, must also be stored in the database so that they may be accessed whenever the BCAM environment is re-started after a shutdown.

6.3 Alternate Storage Formats

Development will be performed to allow the BCAM Environment to store information in text files or other databases than Ingres.

6.4 Installation in Integrated Circuit Fabrication Facilities

The connections to the physical equipment must be implemented. The Berkeley Process-Flow Language[14] (BPFL) and the SECS II protocol will be instrumental in this development.

The BCAM Environment will be installed at the DEC manufacturing facility in Boston, Massachusetts.

6.5 Process Capability (C_{pk}) Evaluation and Process Simulation

When the process prediction capability of the controller is combined with an analysis of the measured standard errors of the equipment (as stored in the equipment models), the controller will be able to determine the total process expected standard error (including any errors which propagate along the process and for which feed-forward control cannot compensate), and thus the controller will be able to determine the expected yield of the process. Hence, whenever a new product is proposed, automated yield prediction for the proposed product may determine whether it is technologically or financially feasible.

6.5.1 Comparison of Control methods

C_{pk} prediction and process simulation can also be used to evaluate the control system itself. By simulating various process problems such as equipment drift or maintenance disturbances, the response of the control system can be examined, and this information can be used to adjust various settings of the controller or even to change the configuration of the process line itself. For example, an engineer may wish to use process simulation to compare the controller responses with feed-forward control to the responses without feed-forward control, and thereby evaluate the benefit of the added control. This simulation may also be used to set forgetting factors for the model update algorithms.

6.5.2 Alarm Generation through C_{pk} Prediction

C_{pk} prediction, when used during production, may also provide a method of alarm generation. When a predicted C_{pk} becomes too low, it can be a signal that new recipes should be created for the equipment, or that some equipment needs servicing.

Appendix A Acronyms

A.1 Acronyms

BCAM	Berkeley Computer Aided Manufacturing
BPFL	Berkeley Process Flow Language
CIM	Computer Integrated Manufacturing
CL	Center Line
CUSUM	CUmulative SUMmation
DEC	Digital Equipment Corporation
EECS	Electrical Engineering & Computer Sciences
EVOP	EVolutionary OPeration
ERL	(Berkeley) Electronics Research Laboratory
ISMSS	International Semiconductor Manufacturing Science Symposium
LCL	Lower Control Limit
LPCVD	Low Pressure Chemical Vapor Deposition
LSL	Lower Specification Limit
SECS	SEMI Equipment Communications Standard
SPC	Statistical Process Control
SRC	Semiconductor Research Corporation
UCB	University of California at Berkeley
UCL	Upper Control Limit
USL	Upper Specification Limit
WIP	Work In Progress

Appendix B Symbols

In this report bold-faced capital letters are used for matrices and bold faced lower case letters for column arrays. Row arrays are obtained by applying the transpose operator (T) to a column array.

$\varnothing$	The set containing only the zero vector.
a	A unitless empirical quantity used in the model update algorithm.
$\boldsymbol{A}$	A derivative matrix of a function f evaluated at a specific point in an input space.
α	An empirical constant used in an Armijo step size calculation.
$\boldsymbol{B}$	A matrix of normalized eigenvectors.
β	An empirical constant used in an Armijo step size calculation.
c	A term coefficient value in a model equation.
Δc	A correction to a term coefficient value in a model equation.
$\boldsymbol{c}$	A vector of term coefficient values in a model equation (not including the constant term).
$\Delta \boldsymbol{c}$	A correction to a vector of term coefficient values in a model equation (not including the constant term).
$\boldsymbol{D}$	A diagonal matrix used to numerically condition vectors of term values used in a model update algorithm.
f	A function mapping an input space into $\Re$.
$\boldsymbol{f}$	A function mapping an input space into an output space.
F	A cost function used in an recipe update algorithm.
g	A linear approximation to a function mapping an input space into $\Re$.
$\boldsymbol{g}$	A linear approximation to a function mapping an input space into an output space.
G	An approximation to a cost function used in a recipe update algorithm.
$\boldsymbol{h}$	A search direction used in a recipe update algorithm.
γ	A vector of transformed term coefficient values.
$\Delta\gamma$	A correction to a vector of transformed term coefficient values.
i	An index of an equipment input setting as in x_i.
j	An index for an output value.
k	An index into a set of data used for regression, OR an iteration index for an optimization algorithm.
κ	The number of data points in a data set used for regression.
l	An index for a term coefficient value, OR an index for an Armijo step size calculation.

L	A linear operator.
λ	An eigenvalue of a variance-covariance matrix.
Λ	A diagonal matrix of eigenvalues.
m	The dimension of an output space.
n	The dimension of an input space.
N	The set of nonnegative integers.
$\mathcal{N}$	A null space of a linear operator.
r	The magnitude of the range of values that an output may take.
$\boldsymbol{R}$	A diagonal matrix used to transform recipes of input settings.
$\mathcal{R}$	The range space of a linear operator.
$\Re$	The set of real numbers.
s	A scaling coefficient used to transform an output value.
$\boldsymbol{S}$	A variance-covariance matrix used in the model update algorithm, or a diagonal matrix used to transform vectors of output values in the recipe update algorithm.
σ	The standard deviation of a random variable.
t	The number of terms (not including the constant term) in a model equation.
$\boldsymbol{t}$	A vector of term values for a model equation.
$\boldsymbol{T}$	A matrix, each row of which is a vector of term values for a model equation.
θ	The zero vector.
$\boldsymbol{u}$	A vector, each element of which is 1.0.
$\boldsymbol{v}$	A numerically conditioned vector of term values for a model equation.
$\boldsymbol{V}$	A matrix, each row of which is a numerically conditioned vector of term values for a model equation.
w	A weighting coefficient for a regression algorithm.
$\boldsymbol{W}$	A diagonal matrix containing the weighting coefficients for a regression algorithm.
x	An equipment input setting.
$\boldsymbol{x}$	A vector (recipe) of equipment settings.
$\Delta\boldsymbol{x}$	A correction to a recipe of equipment settings in a recipe update algorithm.
X	An input space (the set of valid recipes for a machine or a workcell).
$\boldsymbol{X}$	A matrix, each row of which corresponds to a set of input settings.
ξ	A vector in a linear space.
y	A transformed recipe of input settings.
z	An output value.
$\boldsymbol{z}$	A vector of output values.

δz	A difference of a predicted set of output values and a set of desired output values.
Δz	A discrepancy between a measured output value and a predicted output value.
$\Delta \mathbf{z}$	A vector of output discrepancy values.
Z	An output space (the set of possible output value vectors).

Appendix C

Adjoint Derivations of the Recipe Update Equations

Consider the linear operator $L : X \rightarrow Z$, where $L(\boldsymbol{x}) \equiv A\boldsymbol{x}, A \in \Re^{m \times n}$. Then the adjoint operator $L^* : Z \rightarrow X$ is given by $L^*(z) \equiv A^T z, A^T \in \Re^{n \times m}$. These operators satisfy $\langle z, L(\boldsymbol{x}) \rangle = \langle L^*(z), \boldsymbol{x} \rangle, \forall \boldsymbol{x} \in X, \forall z \in Z$.

Denote the null spaces and range spaces of L and L^* by

$$\mathcal{N}_L = \{\boldsymbol{x} \mid L(\boldsymbol{x}) = 0\} \subset \Re^n \tag{38}$$

$$\mathcal{R}_L = \{z \mid z = L(\boldsymbol{x}), \boldsymbol{x} \in \Re^n\} \subset \Re^m \tag{39}$$

$$\mathcal{N}_{L^*} = \{z \mid L^*(z) = 0\} \subset \Re^m \tag{40}$$

$$\mathcal{R}_{L^*} = \{\boldsymbol{x} \mid \boldsymbol{x} = L^*(z), z \in \Re^m\} \subset \Re^n \tag{41}$$

From the properties of the adjoint [15],

$$\mathcal{N}_L \perp \mathcal{R}_{L^*}, \mathcal{N}_L \oplus \mathcal{R}_{L^*} = \Re^n, \tag{42}$$

$$\mathcal{N}_{L^*} \perp \mathcal{R}_L, \mathcal{N}_{L^*} \oplus \mathcal{R}_L = \Re^m, \tag{43}$$

and both $L : \mathcal{R}_{L^*} \rightarrow \mathcal{R}_L$ and $L^* : \mathcal{R}_L \rightarrow \mathcal{R}_{L^*}$ are one-to-one onto mappings. Thus any vector $\boldsymbol{x} \in \Re^n$ can be expressed as

$$\boldsymbol{x} = \boldsymbol{x}' + \boldsymbol{x}'', \tag{44}$$

where $\boldsymbol{x}' \in \mathcal{R}_{L^*}$ and $\boldsymbol{x}'' \in \mathcal{N}_L$, and any vector $z \in \Re^m$ can be expressed as

$$z = z' + z'', \tag{45}$$

where $z' \in \mathcal{R}_L$ and $z'' \in \mathcal{N}_{L^*}$.

Now examining an optimization problem, given is a $\hat{z} \in \Re^m$ and it is desired to find the $\hat{x} \in \Re^n$ such that $\|L(\hat{x}) - \hat{z}\|^2$ is minimized.

Consider the case when $m > n$. Then the system is overdetermined. It is assumed that $\mathcal{N}_L = \varnothing$, which implies that $[A^T A]$ is invertible. From (45), the optimum solution $\hat{x}$ satisfies $A\hat{x} = \hat{z}'$, and $L^T(\hat{z}) = A^T\hat{z} = A^T\hat{z}' + A^T\hat{z}'' = A^T\hat{z}'$. Thus,

$$A^T A\hat{x} = A^T\hat{z}' = A^T\hat{z}, \text{ and} \tag{46}$$

$$\hat{x} = (A^T A)^{-1} A^T\hat{z}. \tag{47}$$

If $m < n$, the system is underdetermined. It is assumed that $\mathcal{N}_{L^*} = \varnothing$, which implies that $[AA^T]$ is invertible and that there is a set of solutions of dimension $n - m$. Suppose that a solution $\hat{x}$ is required which minimizes $\|\hat{x}\|^2$. From (44),

$$L(\hat{x}) = A\hat{x} = A\hat{x}' + A\hat{x}'' = A\hat{x}' = \hat{z}, \tag{48}$$

so that using (42), the optimum solution $\hat{x}$ satisfies $\hat{x}'' = \theta$ (the zero vector), i.e. $\hat{x} \in R_{L^*}$. Therefore it must be of the form $\hat{x} = A^T\xi$, for some $\xi \in \Re^m$. Plugging into (48), $AA^T\xi = \hat{z}$ so that $\xi = (AA^T)^{-1}\hat{z}$, which yields

$$\hat{x} = A^T(AA^T)^{-1}\hat{z}. \tag{49}$$

Now by translating and scaling the coordinate systems, the equations (47) and (49) lead to equations (32) and (31), respectively.

Appendix D Ingres Table Formats

D.1 Tables describing the Ingres tables used by the BCAM environment.

This appendix describes the structure and contents of the tables used by the BCAM environment. The table structures are first described by listing their column names and the types of data stored in those columns. These table structure descriptions are identical to those provided by the Ingres database manager. The meanings of those columns are then briefly described, and an example table is provided.

The data type abbreviations used in the table structure descriptions translate as follows:

- i4 Four byte integer.
- f8 Eight byte floating point number.
- varchar(*n*) A variable length character string of maximum length *n*.
- date A date and time.

D.2 The Equipment Index Table

The 'equipment_index' table basically lists the equipment which is available to the operator. This table is used to create the BCAM environment *Equipment Activation* Window.

Table 1: Ingres table information for 'equipment_index'

Column Name	Data Type	Key #	Nulls	Defaults
name	varchar(20)		yes	n/a
formal_name	varchar(50)		yes	n/a
process_description	varchar(100)		yes	n/a

The column 'name' contains abbreviated names of all available equipment. Each abbreviated name is also used as an affix to the table names of the other Ingres tables which apply to that particular piece of equipment (e.g. the MTI Photoresist Developer's

abbreviated name is 'mti1,' and the name of the table describing it's recipes is 'mti1_recipe.' The column 'formal_name' contains the full names of the equipment, and the column 'process_description' contains descriptions of the functionality of the equipment.

Table 2: Contents of 'equipment_index' database table

name	formal_name	process_description
tylan16	Tylan16 Wafer Furnace	LPCVD of undoped polysilicon
lam1	Lam Autoetch	Plasma etching of polysilicon
eaton	Eaton Photoresist Spinner	Photoresist spinning
mti1	MTI Photoresist Developer	Photoresist develop
gca	GCA Photoresist Exposure	Photoresist exposure

D.3 The Equipment Input/Output Table

The '*equipment*_io' table describes the inputs (settings, controls, or incoming measurements) and outputs (post-processing measurements) of a machine.

Table 3: Ingres table information for 'equipment_io'

Column Name	Data Type	Key #	Nulls	Defaults
parameter	varchar(20)		yes	n/a
abbreviation	varchar(10)		yes	n/a
units	varchar(10)		yes	n/a
io	varchar(10)		yes	n/a
minimum	f8		yes	n/a
maximum	f8		yes	n/a

- parameter — The name of an input or output.
- abbreviation — The abbreviation of the name of an input or output.
- units — The unit of measure for an input or output.
- io — The string 'input' or 'output' to distinguish the two.

- minimum — The minimum valid value for an input or the minimum expected value for an output.
- maximum — The maximum valid value for an input or the maximum expected value for an output.

Table 4: Example of 'mti1_io' table

parameter	abbreviation	units	io	minimum	maximum
thickness	T	Angstroms	input	11000	15000
reflectance	R	%	input	60	90
develop time	DEVTI	seconds	input	50	225
CD1	CD1	microns	output	0.65	1.05

D.4 The Equipment Recipe Table

A '*equipment*_recipe' table contains recipes for the operation of a particular machine.

Table 5: Ingres information on 'equipment_recipe'

Column Name	Data Type	Key #	Nulls	Defaults
step	i4		yes	n/a
critical	i4		yes	n/a
parameter	varchar(20)		yes	n/a
value	f8		yes	n/a
recipe_name	varchar(50)		yes	n/a
recipe_owner	varchar(20)		yes	n/a
date	date		yes	n/a

- step — The operation step (for multistep operation).
- critical — A boolean value; one symbolizes true, and zero symbolizes false. True if this input at this step is critical to values of the machine's outputs. Each input can only be critical at a single step.
- parameter — The name of this input.
- value — The value of this input at this step.
- recipe_name — The name of this recipe.
- recipe_owner — The owner of this recipe.

- date — The date and time at which this recipe was last modified.

Table 6: Example of 'mti1_recipe' table

step	critical	parameter	value	recipe_name	recipe_owner	date[a]
1	1	thickness	13000	default	bcam	19-mar-1992
1	1	reflectance	75	default	bcam	19-mar-1992
1	1	develop time	90	default	bcam	19-mar-1992

a. Although not shown here, the 'date' column also includes a time of day.

D.5 The Equipment Model Structure Table

The '*equipment*_model_structure' table describes the terms which are used in the output models for this machine, i.e. the ways that the inputs can affect the outputs. If there are any transformations on the outputs, they are also described in this table.

Table 7: Ingres information on 'equipment_model_structure'

Column Name	Data Type	Key #	Nulls	Defaults
term	i4		yes	n/a
parameter	varchar(20)		yes	n/a
exp_matrix	f8		yes	n/a
special_function	varchar(10)		yes	n/a

- term — The term name (all term names are required to be positive integers). A negative term name indicates that this row describes an output transformation.
- parameter — The name of an input which shall appear in this term or the name of an output which is being transformed.
- exp_matrix — An exponent to which this parameter will be raised.
- special_function — A special transformation function for this parameter. These special functions are applied to a parameter before taking it to the power of its exponent.

Terms which include more than one input parameter are described by multiple lines in the table.

Table 8: Example of 'tylan16_model_structure' table

term	parameter	exp_matrix	special_function		
-1	thickness	1.0	exp		$Th = e^x$
1	temperature	1.0	none		T
2	time	1.0	none		t
3	silane flow	-1.0	none		$\frac{1}{Q}$
4	temperature	1.0	none		
4	silane flow	-1.0	none		$\frac{T}{Q}$
5	pressure	1.0	none		
5	silane flow	-1.0	none		$\frac{P}{Q}$
6	pressure	1.0	none		
6	temperature	1.0	none		
6	silane flow	-1.0	none		$\frac{P \cdot T}{Q}$
7	silane flow	-1.0	none		
7	time	1.0	none		$\frac{t}{Q}$
8	pressure	1.0	none		
8	silane flow	-1.0	none		
8	time	1.0	none		$\frac{P \cdot t}{Q}$
9	silane flow	-2.0	none		$\frac{1}{Q^2}$
10	temperature	1.0	none		
10	silane flow	-2.0	none		$\frac{T}{Q^2}$

Table 8: Example of 'tylan16_model_structure' table

term	parameter	exp_matrix	special_function		
11	temperature	2.0	none		
11	silane flow	-2.0	none		$\frac{T^2}{Q^2}$
12	time	2.0	none		t^2
13	pressure	1.0	log		$\log P$
14	temperature	-1.0	none		$\frac{1}{T}$
15	time	1.0	log		$\log t$

D.6 The Equipment Models Table

The 'equipment_models' table contains for each machine output the coefficients for terms described in the 'equipment_model_structure' to make up the model for that output.

Table 9: Ingres information on 'equipment_models'

Column Name	Data Type	Key #	Nulls	Defaults
output	varchar(20)		yes	n/a
term	varchar(10)		yes	n/a
coefficient	f8		yes	n/a
model_name	varchar(50)		yes	n/a
model_owner	varchar(20)		yes	n/a
date	date		yes	n/a

- output — The output parameter name.
- term — The term name.
  The term name '0' (zero) is reserved for the constant term of the model.
  The term name 'std err' refers to the estimated gaussian standard error of the machine for this output.
  The term name 'pred err' refers to the estimated prediction error of the output model.
  The term name 'forget fct' refers to the forgetting factor used by this model for model updates.

The term name 'window siz' refers to the maximum window size used by this model for the model updates.

- coefficient — The coefficient value for this term.
- model_name — The name of this model.
- model_owner — The owner of this model.
- date — The most recent modification date of this coefficient.

Table 10: Example of 'tylan16_models' table

Output	Term	Coefficient	Model Name	Model Owner	Date[a]
thickness	window siz	15	default	beam	07-apr-1992
thickness	forget fct	0.95	default	beam	07-apr-1992
thickness	pred err	0.026	default	beam	07-apr-1992
thickness	std err	0.058	default	beam	07-apr-1992
thickness	0	20.65	default	beam	07-apr-1992
thickness	3	-47.97	default	beam	07-apr-1992
thickness	13	0.29	default	beam	07-apr-1992
thickness	14	-15189.2	default	beam	07-apr-1992
thickness	15	1.0	default	beam	07-apr-1992
stress	window siz	15	default	beam	07-apr-1992
stress	forget fct	0.95	default	beam	07-apr-1992
stress	pred err	0.579	default	beam	07-apr-1992
stress	std err	1.261	default	beam	07-apr-1992
stress	0	126.71	default	beam	07-apr-1992
stress	1	-0.172	default	beam	07-apr-1992
stress	2	0.532	default	beam	07-apr-1992
stress	3	-42108	default	beam	07-apr-1992
stress	4	44.8	default	beam	07-apr-1992
stress	5	105.27	default	beam	07-apr-1992
stress	6	-0.112	default	beam	07-apr-1992
stress	7	15.644	default	beam	07-apr-1992
stress	8	-0.039	default	beam	07-apr-1992

Table 10: Example of 'tylan16_models' table

Output	Term	Coefficient	Model Name	Model Owner	Date[a]
stress	9	2725650	default	bcam	07-apr-1992
stress	10	-6070.5	default	bcam	07-apr-1992
stress	11	3.38	default	bcam	07-apr-1992
stress	12	-0.003	default	bcam	07-apr-1992

a. Although not shown here, this column also contains the time of day.

The equations represented by the above tables 'tylan16_model_structure' and 'tylan16_models' are:

$$Th = exp\left(20.65 - \frac{47.97}{Q} + 0.29\log P - \frac{15189.2}{T} + \log t\right) \tag{50}$$

$$S = 126.71 - 0.172T + 0.532t - \frac{42108}{Q} + \frac{44.8T}{Q} + \frac{105.27P}{Q} \ldots$$
$$\ldots - \frac{0.112PT}{Q} + \frac{15.644t}{Q} - \frac{0.039Pt}{Q} + \frac{2725650}{Q^2} - \frac{6070.5T}{Q^2} + \frac{3.38T^2}{Q^2} - 0.003t^2 \tag{51}$$

Appendix E Alphabetical Library Function Listing

The following is an alphabetical list of the C++ functions used in the BCAM environment. After each function is listed its source file name. All source files are located in the directory "~bcamdev/BCAM/bombay-code", except the files "diagnosis.cc" and "recipeIngres.scc" which are located in the directory "bcamdev/BCAM/hcliu-code".

- choiceList choiceClass::AddChoice(char* textPtr) "modelChoice.cc"
- choiceList choiceClass::AddChoiceAlpha(char* textPtr) "modelChoice.cc"
- choiceList choiceClass::AddChoiceToEnd(char* textPtr) "modelChoice.cc"
- void AddEquipment(Widget w, equipNameClass* equipNamePtr, caddr_t call_data) "equipSetup.cc"
- Widget AddIoWidget(Widget io_box, int ioNum, char* text, machineAndIntPtrsType* ioTargetPtr) "connections.cc"
- machineList machineClass::AddMachine(char* equipName) "machineRW.cc"
- void inputClass::AddStep(int step, double value, boolean critical) "steps.cc"
- Widget AddTextWidget(Widget parent, Widget last_widget, char* interrogative, char* information, char* units) "connections.cc"
- void machineClass::AddToHistory() "machineRW.cc"
- boolean machineClass::AreThereConnections() "machineRW.cc"
- wdouble ArmijoStepSize(double alpha, double beta, int peggedInputPtr, double initialCost, double initialDirectionalDerivative, double* searchDirection, double* inputs, connectionClass* processInputs, int inputCount, connectionClass* processOutputs, int outputCount, processNode* processPtr, double* desiredOutputs, double* specRanges) "generateRecipe.cc"
- void AskForFinalSpecs(Widget parent, processList process, caddr_t call_data) "process.cc"
- void AskForFinalTargets(Widget parent, processList process, caddr_t call_data) "process.cc"
- void AskForOutputs(processNode* processPtr, Widget parent) "interfaces.cc"
- void AskForRecipe(Widget parent, processList process, caddr_t call_data) "process.cc"
- void AskForSimulationRepCnt(Widget parent, processList process, caddr_t call_data) "process.cc"
- void processNode::AssignControls(int waferNum, char* interfaceType) "interfaces.cc"
- double Average(double data[], int dataCnt) "mymath.cc"
- void BCAMentrance(int argc, char** argv) "controllerEntrance.cc"
- void BCAMhelp(Widget w, caddr_t client_data, caddr_t call_data) "controllerEntrance.cc"
- void BCAMinfo(Widget w, caddr_t client_data, caddr_t call_data) "controllerEntrance.cc"
- void BcamQuit(Widget w, caddr_t client_data, caddr_t call_data) "controllerEntrance.cc"
- void BCAMXtInitialize(int argc, char** argv) "controllerEntrance.cc"
- void bsubs(double matrix[][MAXDIM], double column[], int dim, short* good) "mymath.cc"
- void processNode::CalcControls() "generateRecipe.cc"
- void CallConnect(Widget w, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"
- void CallDisconnect(Widget w, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"
- void CallGetOutputs(Widget w, processNode* processPtr, caddr_t call_data) "interfaces.cc"
- void CallOldConnect(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void CallPropagate(Widget w, processNode* processPtr, caddr_t call_data) "interfaces.cc"
- void CallRemoveAsGoal(Widget w, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"

- void CallSetAsGoal(Widget w, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"
- char* capitalize(char* string) "machineRW.cc"
- void Center(double data[][MAXDIM], int dim, int dataCnt, double center[][MAXDIM]) "mymath.cc"
- double cerf_prm(double y) "mymath.cc"
- double chebyshev(double a, double b, int n, int i) "mymath.cc"
- void CheckModel(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- int choiceClass::ChoiceListLength() "modelChoice.cc"
- void machineClass::ClearHistory() "machineRW.cc"
- void processNode::ClearHistory() "process.cc"
- void ClearProcessHistory(Widget parent, processList process, caddr_t call_data) "process.cc"
- void ClearProcessHistoryWarning(Widget parent, processList process, caddr_t call_data) "process.cc"
- double outputClass::Coeff(int coeffNum) "modelEval.cc"
- boolean CompatibleUnits(char* units1, char* units2) "machineRW.cc"
- wvoid ConfirmModelCoeffs(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void ConfirmRecipe(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void ConfirmSimulCoeffs(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- boolean inputClass::Connected() "machineRW.cc"
- boolean outputClass::Connected() "machineRW.cc"
- void ConnectIngres(char* databaseName) "modelIngres.scc"
- void ConnectMachines(machineClass* inputMachinePtr, int inputNum, machineClass* outputMachinePtr, int outputNum) "machineRW.cc"
- void ConnectMachines(machineClass* inputMachinePtr, int inputNum, int step, machineClass* outputMachinePtr, int outputNum) "machineRW.cc"
- void machineClass::ConnectMachinesByName(char* inputMachineName, int inputNum, int step, char* outputMachineName, int outputNum) "machineRW.cc"
- void machineClass::CopyOutputsToInputs() "machineRW.cc"
- void processNode::CopyOutputsToInputs() "machineRW.cc"
- double* CopyVector(double* target, double* source, int dimension) "mymath.cc"
- void CopyWidgetLabel(Widget w, Widget target, caddr_t call_data) "modelChoice.cc"
- double outputClass::CorrCoeff(int coeffNum) "modelEval.cc"
- double cosine(double x) "mymath.cc"
- int inputClass::CountSteps() "steps.cc"
- int modelClass::CountTotalInputs() "steps.cc"
- void CovMatrix(double data[][MAXDIM], int dim, int dataCnt, double result[][MAXDIM], double weights[]) "mymath.cc"
- wdouble inputClass::CriticalValue() "steps.cc"
- void CreateMainBCAMmenu() "controllerEntrance.cc"
- inputStepValueClass* inputClass::CriticalStep() "steps.cc"
- int inputClass::CriticalStepNum() "steps.cc"
- processList DeduceProcessFromConnections(machineList machines) "process.cc"
- void choiceClass::DeleteChoiceList() "modelChoice.cc"
- void choiceClass::DeleteChoiceListAndText() "modelChoice.cc"
- void modelNameClass::DeleteModelNameList() "modelRW.cc"
- machineList machineClass::DeleteMachine(machineClass* doomedMachinePtr) "machineRW.cc"
- processNode* processNode::DeleteProcess() "process.cc"
- void recipeNameClass::DeleteRecipeNameList() "recipeIngres.scc"
- void inputClass::DeleteValueList() "steps.cc"

- void DeleteWidgetList(widgetList widgets) "connections.cc"
- int DependenceLength(machineClass* machinePtr, int recursionLevel) "process.cc"
- void Desensitize(Widget w, Widget numb, caddr_t call_data) "functions.cc"
- void DestroyParentOfParent(Widget w, caddr_t client_data, caddr_t call_data) "functions.cc"
- void DestroyParentOfPofPofP(Widget w, caddr_t client_data, caddr_t call_data) "functions.cc"
- void DestroyWidget(Widget w, Widget doomed, caddr_t call_data) "functions.cc"
- double determinant(double M[][MAXDIM], int dim) "mymath.cc"
- void Diagnosis(Widget w, machineClass* machinePtr, caddr_t call_data) "diagnosis.cc"
- void DiagQuit(Widget w, mahcneClass* machinePtr, caddr_t call_data) "diagnosis.cc"
- Widget DialogBelowRight(Widget parent, Widget above, Widget left, char* title,char* information, int vertDist) "functions.cc"
- void DictateControls(processNode* processPtr, Widget parent) "interfaces.cc"
- void DisconnectIngres() "modelIngres.scc"
- void machineClass::DisconnectInput(int inputNum, int step) "machineRW.cc"
- void DisconnectMachines(machineClass* inputMachinePtr, int inputNum, machineClass* outputMachinePtr, int outputNum) "machineRW.cc"
- void DisconnectMachines(machineClass* inputMachinePtr, int inputNum, int step, machineClass* outputMachinePtr, int outputNum) "machineRW.cc"
- void machineClass::DisconnectOutput(int outputNum) "machineRW.cc"
- Widget DisplayInputs(machineClass* machine, Widget equip_form, Widget last_widget) "equipWindow.cc"
- Widget DisplayOutputs(machineClass* machine, Widget equip_form, Widget last_widget) "equipWindow.cc"
- Widget DisplayTerm(machineClass* machine, Widget equip_form, Widget last_widget, Widget left, int termNum, int vertDist) "equipWindow.cc"
- Widget DisplayModel(machineClass* machine, Widget equip_form, Widget last_widget) "equipWindow.cc"
- void DisplayOrigCoeffs(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void DoCalcControls(Widget w, processList process, caddr_t call_data) "process.cc"
- wdouble DotProduct(double* vector1, double* vector2, int dimension) "mymath.cc"
- void dumpInputs(modelClass* modelPtr) "generateRecipe.cc"
- void eigen(double matrix[][MAXDIM], double eigenvalues[], double eigenvectors[][MAXDIM], int dim) "mymath.cc".
- void EquipPredict(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- double erf_prm(double x) "mymath.cc"
- double modelClass::eval(int outputNum, boolean useSimulCoeffs) "modelEval.cc"
- void ExitProcess(Widget parent, processList* processListPtr, caddr_t call_data) "process.cc"
- double exp_div_10000(double_input) "modelSpecFn.cc"
- double exp_div_10000_inv(double input) "modelSpecFn.cc"
- double exp_div_10000_deriv(double input) "modelSpecFn.cc"
- double exp_div_10000_deriv2(double input) "modelSpecFn.cc"
- int factor(int k) "mymath.cc"
- void fsubs(double matrix[][MAXDIM], double column[], int dim) "mymath.cc"
- double Gauss() "modelEval.cc"
- double GetDiscrepancies(double* discrepancies, connectionClass* processOutputs, int outputCount, processNode* processPtr, double desiredOutputs, double* specRanges) "generateRecipe.cc"
- int GetFinalOutputs(connectionClass finalOutputs[], machineList machines) "process.cc"
- int machineClass::GetInputIndex(char* name) "modelEval.cc"
- int machineClass::GetModelIO() "modelIngres.scc"
- int machineClass::GetModelStructure() "modelIngres.scc"

- int machineClass::GetModelCoeffs(char* modelSelector) "modelIngres.scc"
- void GetNewModelName(Widget w, machineClass* machinePtr, caddr_t call_data) "modelRW.cc"
- void GetNewRecipeName(Widget w, machineClass* machinePtr, caddr_t calld_data) "recipeIngres.scc"
- int machineClass::GetOutputIndex(char* name) "modelEval.cc"
- void processNode::GetOutputs() "interfaces.cc"
- int processNode::GetRemainingControls(connectionClass* processInputs) "process.cc"
- int processNode::GetRemainingFinalOutputs(connectionClass* processOutputs) "process.cc"
- int GetSpecialFunctionIndex(char* name) "modelSpecFn.cc"
- int modelClass::GetTermIndex(char* name) "modelEval.cc"
- int modelClass::GetTermIndex(char* name) "modelEval.cc"
- double* modelClass::Gradient(double* gradient, int outputNum) "generateRecipe.cc"
- double* processNode::Gradient(double* gradient, connectionClass* processInputs, int inputCnt, connectionClass processOutput) "generateRecipe.cc"
- void machineClass::Graph(char* graphType) "graphsBCAM.cc"
- void machineClass::Graph(char* graphType, int index) "graphsBCAM.cc"
- void processNode::Graph(char* graphType) "process.cc"
- void GraphAutocorrelation(machineClass* machinePtr, FILE* tempFile, int outputNum) "graphsBCAM.cc"
- void GraphCUSUM(machineClass* machinePtr, FILE* tempFile, int outputNum) "graphsBCAM.cc"
- void GraphFinalOutputs(Widget parent, processList process, caddr_t call_data) "process.cc"
- void GraphInput(machineClass* machinePtr, FILE* tempFile, int inputNum) "graphsBCAM.cc"
- void GraphMachineNormalizedInputs(machineClass* machinePtr, FILE* tempFile) "graphsBCAM.cc"
- void machineClass::GraphN(char* graphType, int indexCount) "graphsBCAM.cc"
- void GraphOutput(machineClass* machinePtr, FILE* tempFile, int outputNum) "graphsBCAM.cc"
- void GraphOutputs(Widget parent, processList process, caddr_t call_data) "process.cc"
- void GraphNormalizedInputs(Widget parent, processList process, caddr_t call_data) "process.cc"
- void GraphPredictionErrors(machineClass* machinePtr, FILE* tempFile, int outputNum) "graphsBCAM.cc"
- void GraphUnconnectedInputs(Widget parent, processList process, caddr_t call_data) "process.cc"
- void HandleConnectAsFinalOutput(Widget w, machineClass* machinePtr, caddr_t call_data) "connections2.cc"
- void HandleConnectAsUncontrollable(Widget w, machineClass* machinePtr, caddr_t call_data) "connections2.cc"
- void HandleInputClick(Widget w, machineClass* machinePtr, caddr_t call_data) "connections2.cc"
- void HandleIOClick(machineClass* clickedMachinePtr, int clickedIoNum, int clickedStep, boolean clickedIsInput, boolean oneSidedConnectStoredIO) "connections2.cc"
- void HandleOutputClick(Widget w, machineClass* machinePtr, caddr_t call_data) "connections2.cc"
- boolean HasAFinalOutput(machineClass* machinePtr) "process.cc"
- void Highlight0Conn(Widget w, Widget io_menu, caddr_t call_data) "connections.cc"
- void HighlightWidget(Widget w, Widget command, caddr_t call_data) "functions.cc"
- historyClass::historyClass() "machineRW.cc"
- historyClass::~historyClass() "machineRW.cc"
- int machineClass::HistoryLength() "machineRW.cc"
- void Inline(Widget w, Widget diagnosis, caddr_t call_data) "diagnosis.cc"
- Widget IOBelowRight(Widget parent, Widget above, Widget left, char* title, char* message, boolean isInput, int vertDist, machineClass* machinePtr) "equipWindow.cc"
- inputClass::inputClass() "modelEval.cc"
- inputClass::inputClass(inputClass& input) "modelEval.cc"
- inputClass::~inputClass() "modelEval.cc"
- double machineClass::inputRange(int inputNum) "modelEval.cc"

- double machineClass::inputRange(ioNameType inputName) "modelEval.cc"
- inputStepValueClass::inputStepValueClass() "steps.cc"
- int modelClass::InsertInputs(double* inputVector) "modelEval.cc"
- void InvalidateHistory(Widget parent, processList process, caddr_t call_data) "process.cc"
- boolean IsAMember(char** unitList, char* unit) "machineRW.cc"
- Widget LabelBelowRight(Widget parent, Widget above, Widget left, char* title, char* message, int vertDist) "functions.cc"
- boolean LegalInputs(connectionClass* processInputs, int inputCount) "generateRecipe.cc"
- short linear(double matrix[][MAXDIM], double column[], short done, short pivt, int dim) "mymath.cc"
- void linerror(short* good) "mymath.cc"
- int machineClass::LoadAllNewRecipe(char* newRecipeName, char* newRecipeOwner) "recipeIngres.scc"
- void LoadCntrlModelWindow(Widget w, machineClass* machinePtr, caddr_t call_data) "modelRW.cc"
- void LoadDefaultModelCoeffs(machineClass* machinePtr, char* modelSelector) "modelRW.cc"
- void LoadDefaultModelToCntrl(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void LoadDefaultModelToSimul(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void LoadModelWindow(Widget w, char* modelSelector, machineClass* machinePtr) "modelRW.cc"
- int machineClass::LoadNewRecipeValues(char* newRecipeName, char* newRecipeOwner) "recipeIngres.scc"
- void LoadRecipeWindow(Widget w, machineClass* machinePtr, caddr_t calld_data) "recipeIngres.scc"
- void LoadSimulModelWindow(Widget w, machineClass* machinePtr, caddr_t call_data) "modelRW.cc"
- double log_deriv(double input) "modelSpecFn.cc"
- double log_deriv2(double input) "modelSpecFn.cc"
- equipNameList equipNameClass::LookupEquipName(equipNameType equipName) "modelRW.cc"
- int choiceClass::LookupChoiceIndex(char* selection) "modelChoice.cc"
- choiceClass* choiceClass::LookupChoicePtr(int selectNum) "modelChoice.cc"
- char* choiceClass::LookupChoiceText(int selectNum) "modelChoice.cc"
- machineClass* machineClass::LookupMachine(char* equipName) "machineRW.cc"
- processNode* processNode::LookupMachine(machineClass* machine) "process.cc"
- modelNameList modelNameClass::LookupModelName(modelNameType modelName, userNameType modelOwner) "modelRW.cc"
- modelNameList modelNameClass::LookupModelName(equipNameType equipName, modelNameType modelName, userNameType modelOwner) "modelRW.cc"
- void ludec(double matrix[][MAXDIM], double column[], int dim, short* good, short pivt) "mymath.cc"
- machineClass::machineClass(char* equipName) "machineRW.cc"
- int MachineCount(machineList machinePtr) "process.cc"
- boolean processNode::MachineInProcess(machineClass* machinePtr) "process.cc"
- main(int argc, char* argv[]) "BCAM.cc"
- void Maint(Widget w, Widget diagnosis, caddr_t call_data) "diagnosis.cc"
- void MaintRank(Widget w, Widget diagnosis, caddr_t call_data) "diagnosis.cc"
- Widget MakeAddEquipMenu(Widget parent, equipNameList choices) "equipSetup.cc"
- Widget machineClass::MakeEquipWindow(Widget parent) "equipWindow.cc"
- Widget MakeIOMenuWidget(Widget connections_menu, Arg* argList, int argNum, char* io_menu_title, boolean inNotOut, Widget io_title_eqPtr, machineAndIntPtrsType* ioTargetPtr) "connections.cc"
- choiceList modelClass::MakeIOChoices(boolean inNotOut) "modelChoice.cc"
- choiceList machineClass::MakeMachineChoices() "modelChoice.cc"
- Widget MakeMainMenu(Widget parent, char* menuTitle, menuCommandType* menuCommands, int commandCnt) "mainMenu.cc"

- void MakeOldConnectionsMenu(Widget parent, machineClass* machines, machineClass* firstMachinePtr) "connections.cc"
- Widget MakeProcessPulldownBar(Widget parent, processList process) "process.cc"
- Widget MakePulldownBar(Widget parent, char* menuTitle, pulldownCommandType* menuCommands, int commandCnt, int barCommandWidth) "mainMenu.cc"
- Widget MakePulldownMenu(Widget parent, char* menuTitle, menuCommandType* menuCommands, int commandCnt) "mainMenu.cc"
- Widget Make3DCommandButton(Widget parent, char* title, char* label, Widget above, Widget left) "functions.cc"
- void MakeWarning(Widget parent, char* warningString, voidFunctionPtrType callbackFn, void* dataPtr) "makeWarning.cc"
- void MapWidget(Widget w, Widget shell, caddr_t call_data) "functions.cc"
- void matinvert(double matrix[][MAXDIM], int dim) "mymath.cc"
- void matmultip(double left[][MAXDIM], double right[][MAXDIM], double result[][MAXDIM], int m, int n, int l) "mymath.cc"
- double maximum(double num1, double num2) "mymath.cc"
- int maximum(int num1, int num2) "mymath.cc"
- void MessageWidget(char* messageString) "messageWidget.cc"
- double minimum(double num1, double num2) "mymath.cc"
- int minimum(int num1, int num2) "mymath.cc"
- modelClass::modelClass() "modelEval.cc"
- char* modelClass::ModelName() "modelEval.cc"
- char* modelClass::ModelOwner() "modelEval.cc"
- int machineClass::ModelUpdate() "modelUpdate.cc"
- int machineClass::ModelUpdate(int outputNum) "modelUpdate.cc"
- boolean modelClass::NoiseState() "modelEval.cc"
- int modelClass::NonzeroTermCoeffsCnt(int outputNum) "modelEval.cc"
- double* NormalizeVector(double* vector, int dimension) "mymath.cc"
- void OfferToGenerateRecipe(Widget parent, processList process) "process.cc"
- void Online(Widget w, Widget diagnosis, caddr_t call_data) "diagnosis.cc"
- void OpenCntrlModel(Widget w, modelNameClass* modelNamePtr, caddr_t call_data) "modelRW.cc"
- void OpenRecipe(Widget w, recipeNameClass* recipeNamePtr, caddr_t calld_data) "recipeIngres.scc"
- void OpenSimulModel(Widget w, modelNameClass* modelNamePtr, caddr_t call_data) "modelRW.cc"
- void processNode::operate(char* interfaceType) "process.cc"
- void processNode::operateNtimes(char* interfaceType, int waferCnt) "process.cc"
- void OperateProcess(Widget parent, processList process, caddr_t call_data) "process.cc"
- void inputClass::operator=(inputClass& input) "modelEval.cc"
- double outputClass::OrigCoeff(int coeffNum) "modelEval.cc"
- outputClass::outputClass() "modelEval.cc"
- outputClass::~outputClass() "modelEval.cc"
- boolean machineClass::OutputIsConnected(int outputNum) "machineRW.cc"
- double machineClass::outputRange(int outputNum) "modelEval.cc"
- double machineClass::outputRange(ioNameType outputName) "modelEval.cc"
- void pivot(int i, double matrix[][MAXDIM], double column[], int dim, short* good) "mymath.cc"
- void Plot(Widget w, int num, caddr_t call_data) "diagnosis.cc"
- void PlotBelief(Widget w, Widget diagnosis, caddr_t call_data) "diagnosis.cc"
- void PopdownParentOfParent(Widget w, caddr_t client_data, caddr_t call_data) "functions.cc"

- void Popdownshell(Widget w, Widget downshell, caddr_t call_data) "functions.cc"
- void PopupLocationShell(Widget shell, Position x, Position y) "functions.cc"
- void PopupLocationShell(Widget w, Widget shell, Position x, Position y, caddr_t call_data) "functions.cc"
- void PopupMessageShell(char* messageText, Widget parent) "functions.cc"
- void PopupShell(Widget parent, Widget child, caddr_t call_data) "functions.cc"
- void PopupShell (Widget parent, Widget child, caddr_t call_data, XtGrabKind grab_kind) "functions.cc"
- void PopupShellGrabNone(Widget parent, Widget child, caddr_t call_data) "functions.cc"
- double power(double x, int n) "mymath.cc"
- double modelClass::predict(int outputNum) "modelEval.cc"
- double machineClass::predict(int outputNum) "modelEval.cc"
- void processNode::predict() "process.cc"
- double processNode::predict(connectionClass output) "modelEval.cc"
- void PredictProcess(Widget parent, processList process, caddr_t call_data) "process.cc"
- void PrepareToAddEquip(Widget parent, void* garbage, caddr_t call_data) "equipSetup.cc"
- Widget PrintBoldInformation(Widget parent, Widget last_widget, char* message) "functions.cc"
- void choiceClass::PrintChoices() "modelChoice.cc"
- void PrintEquipNames(equipNameList equipNames) "modelRW.cc"
- Widget PrintInformation(Widget parent, Widget last_widget, char* message) "functions.cc"
- Widget PrintInformation(Widget parent, Widget last_widget, char* message,char* title) "functions.cc"
- void PrintMatrix(FILE* outputFile, double matrix[][MAXDIM], int rowCnt, int colCnt) "mymath.cc"
- Widget PrintPermaInformation(Widget parent, Widget last_widget, char* message) "functions.cc"
- void PrintVector(FILE* outputFile, double vector[], int dim) "mymath.cc"
- void ProcessAlarms(Widget parent, processList process, caddr_t call_data) "process.cc"
- void ProcessAutocorrelation(Widget parent, processList process, caddr_t call_data) "process.cc"
- void ProcessCUSUM(Widget parent, processList process, caddr_t call_data) "process.cc"
- void ProcessNoiseHistory(Widget parent, processList process, caddr_t call_data) "process.cc"
- processOptionsClass::processOptionsClass() "process.cc"
- void ProcessSetup(Widget parent, void* garbage, caddr_t call_data) "process.cc"
- void processNode::propagate() "interfaces.cc"
- double PvalueFromTstatistic(double t, int df) "mymath.cc"
- void Quit(Widget w, caddr_t client_data, caddr_t call_data) "functions.cc"
- void QuitWarning(Widget w, caddr_t client_data, caddr_t call_data) "controllerEntrance.cc"
- equipNameList ReadEquipNames() "modelIngres.scc"
- void ReadMatrix(char* fileName, double matrix[][MAXDIM], int rowCnt, int colCnt) "mymath.cc"
- modelNameList ReadModelNames(char* equipName) "modelIngres.scc"
- recipeNameList ReadRecipeNames(char* equipName) "recipeIngres.scc"
- void ReadTextIntoDouble(Widget w, double* target, caddr_t call_data) "modelChoice.cc"
- void ReadTextIntoInputValue(Widget w, inputStepValueClass* target, caddr_t c_d) "modelChoice.cc"
- void ReadTextIntoOutputValue(Widget w, outputClass* target, caddr_t call_data) "modelChoice.cc"
- void ReadTolerance(Widget w, outputClass* target, caddr_t call_data) "modelChoice.cc"
- void ReadVector(char* fileName, double* vector, int dim) "mymath.cc"
- void Regress(double* x, double* y, double* weights, int dim, double* slopePtr, double* pValueSlopePtr, double* interceptPtr, double* pValueIntPtr, double* sigmaSqHatPtr) "mymath.cc"
- void Realize(Widget w, Widget shell, caddr_t call_data) "functions.cc"
- void RememberRepCnt(Widget w, int* repCntPtr, caddr_t call_data) "process.cc"
- char* modelClass::RecipeName() "modelEval.cc"

- char* modelClass::RecipeOwner() "modelEval.cc"
- void RedoIoMenu(Widget io_box, machineAndIntPtrsType* ioTargetPtr, boolean inNotOutMenu) "connections.cc"
- void RedoIoMenus(Widget equiop, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"
- void machineClass::RefreshOutputConnectionMenu(int outputNum) "machineRW.cc"
- void RefreshWidget(Widget w, Widget shell, caddr_t call_data) "functions.cc"
- void machineClass::RemoveAsGoal(int outputNum) "machineRW.cc"
- void machineClass::RemoveAsUncontrollable(int inputNum, int step) "machineRW.cc"
- void RemoveEquipmentWarning(Widget w, machineClass* doomedMachine, caddr_t call_data) "equipSetup.cc"
- void RemoveMachine(Widget w, machineClass* doomedMachine, caddr_t call_data) "equipSetup.cc"
- void Resensitize(Widget w, Widget numb, caddr_t call_data) "functions.cc"
- void RestoreRecipe(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- double root(double x, int n) "mymath.cc"
- int modelClass::SaveCoeffs() "modelIngres.scc"
- int modelClass::SaveModel() "modelRW.cc"
- void SaveModel(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void SaveModelAs(Widget w, machineClass* machinePtr, caddr_t call_data) "modelRW.cc"
- void SaveModelWarning(Widget w, machineClass* machinePtr, caddr_t call_data) "equipWindow.cc"
- void SaveRecipe(Widget w, machineClass* machinePtr, caddr_t calld_data) "recipeIngres.scc"
- void SaveRecipeAs(Widget w, machineClass* machinePtr, caddr_t calld_data) "recipeIngres.scc"
- double modelClass::sens(int outputNum, int inputNum) "modelEval.cc"
- double processNode::sens(connectionClass output, connectionClass input) "modelEval.cc"
- void machineClass::SetAsGoal(int outputNum) "machineRW.cc"
- void machineClass::SetAsUncontrollable(int inputNum, int step) "machineRW.cc"
- void SetChoice(Widget w, returnChoiceType* returnChoicePtr, caddr_t call_data) "modelChoice.cc"
- double outputClass::SetCoeff(int coeffNum, double newValue) "modelEval.cc"
- double outputClass::SetCorrCoeff(int coeffNum, double newValue) "modelEval.cc"
- double inputClass::SetCriticalValue(double newValue) "steps.cc"
- void SetEquip2(Widget w, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"
- void machineClass::SetHistoryToInvalid() "machineRW.cc"
- void processNode::SetHistoryToInvalid() "process.cc"
- void SetIoNum(Widget w, machineAndIntPtrsType* target, caddr_t call_data) "connections.cc"
- void SetIoNumToZero(Widget w, int* ioNumPtr, caddr_t call_data) "connections.cc"
- char* modelClass::SetModelName(char* newName) "modelEval.cc"
- char* modelClass::SetModelOwner(char* newName) "modelEval.cc"
- char* modelClass::SetRecipeName(char* newName) "modelEval.cc"
- char* modelClass::SetRecipeOwner(char* newName) "modelEval.cc"
- double outputClass::SetSimulCoeff(int coeffNum, double newValue) "modelEval.cc"
- char* modelClass::SetSimulModelName(char* newName) "modelEval.cc"
- char* modelClass::SetSimulModelOwner(char* newName) "modelEval.cc"
- void SetTextWidgetString(Widget w, char* string) "functions.cc"
- void SetToFalse(Widget w, int* targetIntPtr, caddr_t call_data) "modelChoice.cc"
- void SetToTrue(Widget w, int* targetIntPtr, caddr_t call_data) "modelChoice.cc"
- double inputClass::SetValue(int step, double newValue) "steps.cc"
- double inputStepValueClass::SetValue(double newValue) "steps.cc"
- double outputClass::SetValue(double newValue) "modelEval.cc"

- void SetWidgetLabel(Widget w, char* label) "functions.cc"
- void ShowConnections(Widget parent, connectionInfoType* connectionInfoPtr, caddr_t call_data) "connections.cc"
- void processNode::ShowProcessOrder(Widget parent, Widget command_bar) "process.cc"
- void processNode::ShowPredictedFinalOutputs(Widget parent) "process.cc"
- double machineClass::simulate() "modelEval.cc"
- double modelClass::simulate(int outputNum) "modelEval.cc"
- void SimulateProcess(Widget parent, processList process, caddr_t call_data) "process.cc"
- void SimulateProcessNtimes(Widget parent, processList process, caddr_t call_data) "process.cc"
- void SimulateProcessNtimes(Widget parent, processList process, caddr_t call_data) "process.cc"
- double outputClass::SimulCoeff(int coeffNum) "modelEval.cc"
- char* modelClass::SimulModelName() "modelEval.cc"
- char* modelClass::SimulModelOwner() "modelEval.cc"
- Widget SizedDialogBelowRight(Widget parent, Widget above, Widget left, char* title,char* information, int size, int vertDist) "functions.cc"
- Widget SizedLabelBelowRight(Widget parent, Widget above, Widget left, char* title,char* message, int size, int vertDist) "functions.cc"
- specialFunctionType spec_fns[SPEC_FN_CNT] "modelSpecFn.cc"
- void StartBCAM(Widget w, Widget bcam, caddr_t call_data) "controllerEntrance.cc"
- inputStepValueClass* inputClass::Step(int step) "steps.cc"
- boolean inputClass::StepExists(int step) "steps.cc"
- char* strip(char* string) "modelIngres.scc"
- double Sum(double data[], int dataCnt) "mymath.cc"
- double SumOfSquares(double data[], int dataCnt) "mymath.cc"
- void svd(double matrix[][MAXDIM], double left[][MAXDIM], double singularValues[], double right[][MAXDIM], int rowCnt, int colCnt) "mymath.cc"
- double modelClass::term(int termNum) "modelEval.cc"
- double modelClass::termDeriv(int termNum, int inputNum) "modelEval.cc"
- double modelClass::termMax(int termNum) "modelEval.cc"
- double modelClass::termMin(int termNum) "modelEval.cc"
- void modelClass::termMinMax(int termNum, int position, double* minPtr, double* maxPtr) "modelEval.cc"
- double modelClass::termRange(int termNum) "modelEval.cc"
- void TimeSeed() "modelEval.cc"
- double Tintegral(double t, int df) "mymath.cc"
- double Tintegral(double t, double ai, double n) "mymath.cc"
- void ToggleFeedback(Widget w, processList process, caddr_t call_data) "process.cc"
- void ToggleFeedForward(Widget w, processList process, caddr_t call_data) "process.cc"
- double Transform(double plain, transformType xform) "modelEval.cc"
- void TransformInputs(double* inputVector, double* inputRanges, int dimension) "generateRecipe.cc"
- void Transpose(double matrix[][MAXDIM], int rows, int columns, double target[][MAXDIM]) "mymath.cc"
- int trunc(double x) "mymath.cc"
- void machineClass::TurnOffFeedback() "machineRW.cc"
- void processNode::TurnOffFeedback() "process.cc"
- void modelClass::TurnOffNoise() "modelEval.cc"
- void processNode::TurnOffNoise() "process.cc"
- void machineClass::TurnOnFeedback() "machineRW.cc"
- void processNode::TurnOnFeedback() "process.cc"

- void modelClass::TurnOnNoise() "modelEval.cc"
- void processNode::TurnOnNoise() "process.cc"
- void UnhighlightWidget(Widget w, Widget command, caddr_t call_data) "functions.cc"
- void UnmapWidget(Widget w, Widget shell, caddr_t call_data) "functions.cc"
- void Unrealize(Widget w, Widget shell, caddr_t call_data) "functions.cc"
- double Untransform(double fancy, transformType xform) "modelEval.cc"
- void UntransformInputs(double* inputVector, double inputRanges, int dimension) "generateRecipe.cc"
- boolean ValidConnection(machineClass* inputMachinePtr, int inputNum, machineClass* outputMachinePtr, int outputNum) "machineRW.cc"
- int machineClass::ValidHistoryLength() "machineRW.cc"
- double inputClass::Value(int step) "steps.cc"
- double inputStepValueClass::Value() "steps.cc"
- double outputClass::Value() "modelEval.cc"
- inputStepValueClass* inputClass::ValueList() "steps.cc"
- double VectorLength(double* vector, int dimension) "mymath.cc"
- double WeightedAverage(double data[], double weights[], int dataCnt) "mymath.cc"
- void modelClass::ZeroCoeffs() "modelEval.cc"
- void modelClass::ZeroCoeffs(int outputNum) "modelEval.cc"
- void ZeroMatrix(matrixType matrix, int rowCnt, int colCnt) "mymath.cc"
- void ZeroVector(vectorType vector, int dim) "mymath.cc"

Appendix F Description of the Source Files

All source files are located in the directory "~bcamdev/BCAM/bombay-code", except the files "diagnosis.cc" and "recipeIngres.scc" which are located in the directory "bcamdev/BCAM/hcliu-code". The source files are accessed through the UNIX revision control system rcs. To check out a file, an authorized user must issue the command

```
co -l filename
```

which will give that user write permission to the file. Once changes have been made to the file, the user must check the file back in using the command

```
ci -u filename
```

To put a new source file under rcs control, the owner must issue the command

```
ci newfilename
```

To add authorization to access a source file for a particular user, the following command must be issued

```
rcs -ausername
```

Note that the user name immediately follows the -a switch. For further details on rcs, reference the UNIX manual pages.

F.1 BCAM.cc The is the location of the *main* function. All the main function does is call the BCAMentrance function. The purpose of this separate main function is to allow that BCAMentrance can in the future be called from higher level software.

- main(int argc, char* argv[])
  {

```
extern void BCAMentrance(int,char**);
BCAMentrance(argc,argv);
}
```

F.2 connections.cc This file contains the code for the old style connections window which is accessed through the Equipment Window's Connections Menu. The code in this file is somewhat outdated; newer code for menu-guided connections should be written in the future.

F.3 connections2.cc This file contains the code for interpreting clicks upon input and output names in the Equipment Window. These clicks are used to infer machine interconnection information.

F.4 controllerEntrance.cc This file contains the functions used to start up the BCAM environment, including the initialization of the X Windows, and the creation of the entrance window and the main BCAM menu. This file also contains the declarations of the global variables of the BCAM environment.

F.5 diagnosis.cc This file contains the functions for accessing the BCAM diagnosis application [16][17]. This source code is written and maintained by Hao-Cheng Liu.

F.6 equipSetup.cc This file contains the higher level functions used to activate and deactivate equipment. With activation (deactivation), an instance of machineClass is created (deleted).

F.7 equipWindow.cc This file contains all the code for creating an Equipment Window, and most of the code for the Equipment Window's callback functions.

F.8 functions.cc General functions (mostly callback) to deal with X windows.

F.9 generateRecipe.cc This file contains the code for calculating equipment setting recipes. Most of this code is mathematical in nature.

F.10 graphsBCAM.cc This file contains code to present graphs for the BCAM environment. These graphs concern themselves with the analysis of measurement records. The functions in this file make use of the unix command 'xgraph'. This file does not contain the source code for response surface plotting of equipment models.

F.11 interfaces.cc This file contains code for using different interfaces to run or simulate wafer processing. This is where code should be inserted for automatic communication to processing equipment and measurement equipment.

F.12 machineRW.cc This file contains a variety of functions dealing with machineClass instances.

F.13 mainMenu.cc This file contains the functions used to build the main BCAM menu and the bars of pulldown menus at the tops of the equipment window and the workcell window.

F.14 makeWarning.cc This file contains the function which creates a BCAM warning window.

F.15 messageWidget.cc This file contains the function to create a BCAM message window.

F.16 modelChoice.cc This file contains functions for creating lists of choices in a widget format. The functions in this file are referenced primarily from those in the 'connections.cc' file. This code is provided primarily for use with the old style connections window.

F.17 modelEval.cc This file contains low level functions for evaluating models, accessing private members of the *modelClass*, and other basic mathematical operations on machines and models. This file also contains constructor and destructor functions for several classes.

F.18 modelIngres.scc This file contains the code for accessing the Ingres table 'equipment_index' as well as the Ingres tables for equipment input/output information and equipment models. This code requires an Ingres pre-processor (esqlc).

F.19 modelRW.cc This file contains functions dealing with the reading and writing of models (i.e. loading them and saving them), as well as functions dealing with the reading of the available equipment list. These functions are one level up from the functions which actually handle the database operations.

F.20 modelSpecFn.cc This file contains the code for implementing several special functions. The needed functions are defined here, as well as an array of pointers to those functions. Another function is used to find indices into this array.

F.21 modelUpdate.cc This file contains the code used for updating equipment models. The model update algorithm uses several mathematical functions defined in "mymath.cc".

F.22 mymath.cc This file contains mathematical functions. Most of the functions in this file were written by Professor Costas Spanos. The statistical analysis functions were added by the author.

F.23 process.cc This file contains most of the functions which handle a processing workcell, including the functions which define the Workcell Window.

F.24 recipeIngres.scc This file contains code for the BCAM Recipe Editor's interface to the Ingres database. This code was written by Hao-Cheng Liu and Bart Bombay and is now maintained by Hao-Cheng Liu. This code requires an Ingres pre-processor (esqlc).

F.25 steps.cc This file contains functions for accessing the recipe values at individual steps in the operation of a machine.

F.26 xgraphRM.cc This file contains a function which creates an xgraph, and then when the xgraph exits, automatically removes the file which was used for that xgraph. This file is self contained (except for the libraries <stdio.h> and <stdlib.h>) and compiles to its own executable.

- ```
  main(int argc, char* argv[])
  {
   if (argc<2) {
   fprintf(stderr,"xgraphRM error: no file name specified!\n");
   exit(0);
   }
   char systemString[250];
   sprintf(systemString,"xgraph -bg white -bw 10 -tk -bb %s",argv[1]);
   system(systemString);
   sprintf(systemString,"rm %s",argv[1]);
   system(systemString);
  }
  ```
  ```

Appendix G Principal Library Functions Listed by Hierarchy

This appendix describes the basic program flow of several important sections of the BCAM program. Function nesting is indicated by the number of bullets preceding a function name; each bullet represents a level of nesting. Only the most important function calls are listed in this description; minor functions calls can be referenced in the source code.

G.1 The Beginning of the Program

- BCAMentrance — Create BCAM entrance window with 'Continue', 'Quit'.
  - BCAMXtInitialize — Initialize X windows for BCAM.
  - XtMainLoop — Loop for events (mouse clicks, keyboard strikes).

G.1.1 Click on Entrance Window: 'Continue'

- StartBcam — Bring up the main BCAM menu.
  - ConnectIngres — Connect to the Ingres database.
  - XtUnrealizeWidget — Bring down the BCAM entrance window.
  - CreateMainBCAMMenu — Define and create the main BCAM menu window and realize it to the screen.
    - MakeMainMenu — Create the main menu command buttons and set up the callbacks for these buttons.

G.1.2 Click on Entrance Window: 'Quit'

- BcamQuit — Shut down the BCAM Environment.

G.2 Main BCAM Menu Actions

G.2.1 Click on 'EQUIP'

- PrepareToAddEquip — Bring up window with equipment activation menu.
  - ReadEquipNames — Read the list of available equipment from the database and store in memory (global linked list *availableEquipment*).
  - MakeAddEquipmentMenu — Define and create the window for equipment activation.
  - XtPopup — Pop up the equipment activation window to the screen.

G.2.2 Click on 'HELP'

- BCAMhelp — Pop up the main BCAM help window.[1]

G.2.3 Click on 'INFO'

- BCAMinfo — Pop up the BCAM information window.

G.2.4 Click on 'EXIT'

- QuitWarning — Ask the user for verification before quitting.
  - BcamQuit — Shut down the BCAM Environment.

G.3 Activation of a Machine

- AddEquipment — Add a machine to the current setup and bring up the corresponding equipment window.
  - AddMachine — Create a new *machineClass* instance and load the machine data from the database.
  - MakeEquipWindow — Define, create, and pop up the equipment window for this machine.

G.3.1 Function AddMachine

- machineClass — Constructor for *machineClass*. Load machine information from the database.
  - LookupEquipName — Look up machine in list of available equipment.
  - GetMachine — Load up machine information from the database.
    - GetModelIO — Get the machine's input and output information.
    - GetModelStructure — Load the machine's model structure from the database.
    - GetDefaultModelCoeffs — Load the machine's model coefficients from the database.
  - LoadAllNewRecipe — Load the machine's default recipe from the database.

G.3.2 Function MakeEquipWindow

- XtCreatePopupShell — Create the window shell for the equipment window.
- MakePulldownBar — Make the bar of pulldown menus along the top of the equipment window.
  - MakePulldownMenu — Make a pulldown menu within the pulldown bar.
- DisplayInputs — Put all the input information in window.
- DisplayOutputs — Put all the output information in window.

---

1. The BCAM environment help facility has not yet been developed.

- DisplayModel — Put all the model information in window.
- PopupshellGrabNone — Pop up the equipment window.
- DisplayModelCoeffs — Set the equipment window to show the controller's current adaptive model.

G.4 Equipment Model Analyses

G.4.1 Controller's Current Adaptive Model Evaluation

- processNode::predict — Predict the outputs of all machines in the workcell. Start with *this* machine, and use current input values.
  - machineClass::predict — Predict an output of *this* machine.
    - modelClass::predict — Evaluate the controller's current model for an output.
      - modelClass::eval — Evaluate the equipment model.
        - modelClass::term — Evaluate a term in the equipment model.

G.4.2 Simulator Model Evaluation

- machineClass::simulate — Simulate all machine outputs using the simulator model.
  - modelClass::simulate — Evaluate the equipment simulator model for an output.
    - modelClass::eval — Evaluate the equipment model.
      - modelClass::term — Evaluate a term in the equipment model.

G.4.3 Controller's Current Adaptive Model Sensitivity Calculation

- processNode::sens — Calculate the sensitivity of one machine's output to another machine's input.
  - modelClass::sens — Calculate the sensitivity of an output to an input. (Evaluate the derivative of the equipment model.)
    - modelClass::termDeriv — Evaluate the derivative of a term in the equipment model.
    - modelClass::eval — Evaluate the equipment model.
      - modelClass::term — Evaluate a term in the equipment model.

G.5 Workcell Operation

G.5.1 Simulating the Operation of the Workcell (for demonstration and debugging purposes)

- SimulateProcess — Workcell window menu command button callback.
  - processNode::operate("simulator") — Operate workcell in simulator mode.

• • • processNode::AssignControls
Assign the current controls to a simulated machine.

• • • • machineClass::simulate
Simulate the operation of a machine.

• • • • processNode::GetOutputs
Fetch the results of the simulation.

• • • • • processNode::propagate
Deal with the results of the simulation.

• • • • • • machineClass::CopyOutputsToInputs
Send the results on to other machine inputs as specified by connections.

• • • • • • machineClass::AddToHistory
Store the results in the machine's historical record.

• • • • • • machineClass::ModelUpdate
Check the machine's model and update if necessary.

• • • • • • processNode::CalcControls
Handle feed forward control calculations.

• • • • • • processNode::AssignControls
If not done, start simulation of the next processing station in the workcell.

• • • • • • processNode::CalcControls
If done with the last machine in the workcell and any models have been updated and feedback control is on, then calculate a new workcell recipe if necessary.

G.5.2 Wafer Processing in the Workcell

• OperateProcess Workcell window menu command button callback.

• • processNode::operate("screen interface")
Operate workcell in screen interface mode.

• • • processNode::AssignControls
Assign the current controls to a machine.

• • • • processNode::DictateControls
Pop up a window dictating the current controls for a machine.

• • • • • CallGetOutputs DictateControls callback function to call processNode::GetOutputs.

• • • • • • processNode::GetOutputs
Fetch the results of the simulation.

• • • • • • • AskForOutputs Pop up a window asking the operator for the results (measurements) after the machine is finished.

• • • • • • • • CallPropagate AskForOutputs callback function to call processNode::propagate.

• • • • • • • • • processNode::propagate
Deal with the results typed in by the operator.

• • • • • • • • • • machineClass::CopyOutputsToInputs
Send the results on to other machine inputs as specified by connections.

• • • • • • • • • • machineClass::AddToHistory
Store the results in the machine's historical record.

• • • • • • • • • • machineClass::ModelUpdate
Check the machine's model and update if necessary.

• • • • • • • • • • processNode::CalcControls
Handle feed forward control calculations.

• • • • • • • • • • processNode::AssignControls
If not done, start operation of the next processing station in the workcell.

• • • • • • • • • • processNode::CalcControls
If done with the last machine in the workcell and any models have been updated and feedback control is on, then calculate a new workcell recipe if necessary.

Appendix H Source Code Listings

Source code for the BCAM Environment is available upon request by electronic mail to bcam@radon.berkeley.edu.

Appendix I Known Bugs

- Unit compatibility is checked when connections are made (e.g. micrometers are compatible with Angstroms). The BCAM Workcell Controller, however, does not support unit conversions along connections. 5/92

Appendix J

A G2 Formulation of Measurement Queuing Effects

J.1 Introduction

Recent developments in integrated circuit design call for the improvement of the performance of photolithography workcells. In order to accomplish this improvement, computer aided manufacturing techniques are being applied to the process, and these techniques require regular measurements of equipment performance. With these measurements, however, are the associated costs of additional hardware, time, and labor. Hence the industry is faced with the problem of implementing these measurements in a manner which will minimize the cost per unit product produced yet improve product quality. The most obvious goals are to increase product yield (decrease the fraction nonconforming) and improve product performance. Because measurements will slow down the manufacturing process, the desired implementation will attempt to minimize the impact of taking these measurements upon the product throughput, and thus attempt to maintain a satisfactory production level.

There are several issues which must be addressed in any formulation of this scheme. Specifications must be determined on how many wafers to measure, which wafers to measure, and how often to measure them. The types of measurements must be decided upon. The effects on work in progress inventory must be examined. And finally the production costs must be studied to determine the magnitude of any improvement in marginal cost versus marginal revenue.

The Berkeley Computer Aided Manufacturing (BCAM) group was recently presented with the opportunity to study a new software product from Gensym Corporation. The product, G2, is a flexible tool which uses an object oriented environment to simulate and

control various types of systems. Of particular interest are this product's extended graphical capabilities which assist an operator in using the system.

For this study, the G2 software was used to simulate a photolithography workcell and to study the effects of introducing a measurement strategy into the workcell. The feasibility of using G2 as an interface to a control and monitoring system is also addressed.

J.2 Methodology

The G2 software possesses several appealing features. Among these are its graphics capabilities, its object oriented environment, its simulation ability, and its general flexibility. In order to introduce customers to the software, Gensym provides a two day course on the G2 system. This course proved effective in familiarizing new users with the general use of G2.

Although G2 is very flexible, considerable effort must be expended to program algorithms into G2. The one second clock cycle for G2 is also rather restrictive. These limitations prevent the use of G2 to implement the generalized BCAM control and monitoring system. However, one interesting feature of the software is its ability to interface with C programs. This feature leads to the possibility of a more limited use of G2 as a graphical interface to the BCAM software, which is written primarily in C++. While such an implementation is attractive, the high cost of G2 precludes such a limited use. For this study a more limited application is chosen, namely the queuing problems associated with introducing regular measurements into a photolithography workcell.

The design for this study focuses on the construction of a relatively simple model of the photolithography workcell timing (see figure below). Wafers are processed by a machine and then placed into a storage area. From this area, wafers are either taken to an analytical station for measurement and then transferred to the following storage area, or

they are transferred directly to the next storage area. The next processing station then takes its wafers from that storage area. The decisions about whether or not to measure any particular wafer are dependent upon production flow and control criteria.

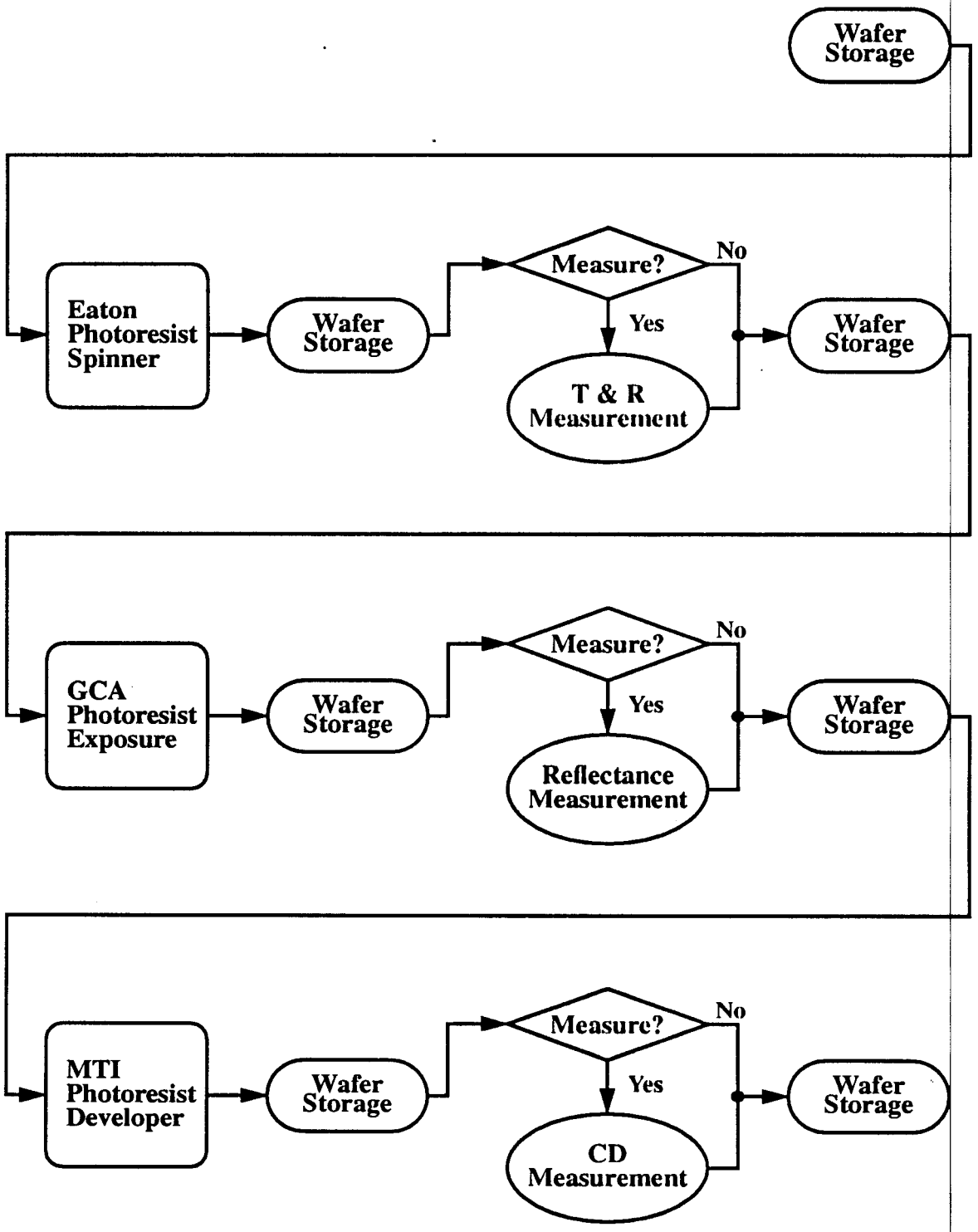

Figure 7 Flow Diagram for the Photolithography Workcell

Initial work to design a knowledge base with G2 includes the definition of several object types and icons, and preliminary connections among instances of these icons. Gensym also provides a customer support visit which is effective in assisting users new to the G2 system. With such assistance, a basic design can be implemented. This basic design may then be further refined with the introduction of an enhanced set of rules, more informative readouts, and more precise timing specifications.

J.3 Results

This study compares two distinct algorithms for the scheduling of wafer measurements. These two methods are henceforth referenced as algorithm **A** and algorithm **B** and are described below.

J.3.1 Wafer Scheduling Algorithm A

The first method, algorithm **A**, uses inventory based rules to decide the number of wafers from which measurements would be taken. Each storage area immediately preceding a processing station has a specific low level. If the wafer count in the storage area falls below this low level, the deficit is immediately taken from the preceding storage area, and the wafers so taken do not get measured. As long as the count of the storage areas remains above or at the low level, each wafer will be subjected to measurement as it passes between storage areas.

Algorithm A proves to be successful in maintaining production levels since it foregoes measurements whenever the relevant intermediate wafer inventories fall below designated low levels. Because wafers require queuing before measurement, this formulation does, however, increase the work in progress inventory. Another drawback to this method is the variability of the frequency of measurement; during some time periods, many wafers are measured, while during other time periods few or no wafers are measured.

J.3.2 Wafer Scheduling Algorithm B

The second algorithm for wafer measurement, algorithm **B**, sets specific goals for the number of wafers to be measured at each step in the process. One out of every four wafers is subjected to measurement as it passes between storage areas. This algorithm is indifferent to the supply levels in the storage areas.

Algorithm B is successful at providing a steady stream of data, but results in a somewhat reduced production level. This method also results in a lower work in progress inventory than the first method, although still higher than a process without measurements.

J.3.3 Results of the Simulations

It should be noted that the simulations fail to give a good quality of information. This deficiency is attributed to two factors. The first limitation of the system is that it discretizes time into one second intervals. Thus if the time were scaled to simulate five minutes of production every second, then the resolution of the process simulation would be limited to five minutes. The time scale chosen for the simulation is one minute per second. This time scale yields sufficient resolution for the simulation, while providing results after a reasonable period of time. (At this scale, the simulation of a 24 hour workday requires 24 minutes.) The second limitation of the system results from the unmanageability of the G2 rule system. Creating complex rule patterns with G2, although undoubtedly possible, is time consuming, particularly with respect to making structural changes in the flow decision rules. Hence time requirements for the implementation of the desired simulations exceed the resources allotted to the project. The project is therefore somewhat scaled back.

Several topics of interest are ignored in these simulations. An analysis of the profitability of the different algorithms is not performed. The simulation of the workcell is